Memoria

La memoria de una computadora almacena los datos de entrada, programas que se han de ejecutar y resultados. En la mayoría de las computadoras existen dos tipos de memoria principal: memoria de acceso aleatorio RAM que soporta almacenamiento temporal de programas y datos y memoria de sólo lectura ROM que almacena datos o programas de modo permanente. La memoria central (RAM, Random, Access Memory) o simplemente memoria se utiliza para almacenar, de modo temporal información, datos y programas. En general, la información almacenada en memoria puede ser de dos tipos: las instrucciones de un programa y los datos con los que operan las instrucciones. Para que un programa se pueda ejecutar (correr, rodar, funcionar..., en inglés run), debe ser situado en la memoria central, en una operación denominada carga (load) del programa. Después, cuando se ejecuta (se realiza, funciona) el programa, cualquier dato a procesar por el programa se debe llevar a la memoria mediante las instrucciones del programa. En la memoria central, hay también datos diversos y espacio de almacenamiento temporal que necesita el programa cuando se ejecuta y así poder funcionar. La memoria principal es la encargada de almacenar los programas y datos que se están ejecutando y su principal característica es que el acceso a los datos o instrucciones desde esta memoria es muy rápido. Es un tipo de memoria volátil (su contenido se pierde cuando se apaga la computadora); esta memoria es, en realidad, la que se suele conocer como memoria principal o de trabajo; en esta memoria se pueden escribir datos y leer de ella. Esta memoria RAM puede ser estática (SRAM) o dinámica (DRAM) según sea el proceso de fabricación. Las memorias RAM actuales más utilizadas son las SDRAM en sus dos tipos: DDR (Double Data Rate) y DDR2.

La memoria ROM, es una memoria que almacena información de modo permanente en la que no se puede escribir ya que es una memoria de sólo lectura. Los programas almacenados en ROM no se pierden al apagar la computadora y cuando se enciende, se lee la información almacenada en esta memoria. Al ser esta memoria de sólo lectura, los programas almacenados en los chips ROM no se pueden modificar y suelen utilizarse para almacenar los programas básicos que sirven para arrancar la computadora. Con el objetivo de que el procesador pueda obtener los datos de la memoria central más rápidamente, la mayoría de los procesadores actuales (muy rápidos) utilizan con frecuencia una memoria denominada caché que sirva para almacenamiento intermedio de datos entre el procesador y la memoria principal. La memoria caché —en la actualidad— se incorpora casi siempre al procesador. Los programas y los datos se almacenan en RAM. La memoria RAM es una memoria muy rápida y limitada en tamaño, sin embargo la computadora tiene otro tipo de memoria denominada memoria secundaria o almacenamiento secundario que puede crecer comparativamente en términos mucho mayores. La memoria secundaria es realmente un dispositivo de almacenamiento masivo de información y por ello, a veces, se la conoce como memoria auxiliar, almacenamiento auxiliar, almacenamiento externo y memoria externa.

DATOS, TIPOS DE DATOS

El primer objetivo de toda computadora es el manejo de la información o datos. Estos datos pueden ser las cifras de ventas de un supermercado o las calificaciones de una clase. Un dato es la expresión general que describe los objetos con los cuales opera una computadora. La mayoría de

las computadoras pueden trabajar con varios tipos (modos) de datos. Los algoritmos y los programas correspondientes operan sobre esos tipos de datos. La acción de las instrucciones ejecutables de las computadoras se refleja en cambios en los valores de las partidas de datos. Los datos de entrada se transforman por el programa, después de las etapas intermedias, en datos de salida. En el proceso de resolución de problemas el diseño de la estructura de datos es tan importante como el diseño del algoritmo y del programa que se basa en el mismo. Un programa de computadora opera sobre datos (almacenados internamente en la memoria almacenados en medios externos como discos, memorias USB, memorias de teléfonos celulares, etc., o bien introducidos desde un dispositivo como un teclado, un escáner o un sensor eléctrico). En los lenguajes de programación los datos deben de ser de un tipo de dato específico. El tipo de datos determina cómo se representan los datos en la computadora y los diferentes procesos que dicha computadora realiza con ellos.

Tipo de datos

Conjunto específico de valores de los datos y un conjunto de operaciones que actúan sobre esos datos Existen dos tipos de datos: básicos, incorporados o integrados (estándar) que se incluyen en los lenguajes de programación; definidos por el programador o por el usuario.

Además de los datos básicos o simples, se pueden construir otros datos a partir de éstos, y se obtienen los datos compuestos o datos agregados, tales como estructuras, uniones, enumeraciones (subrango, como caso particular de las enumeraciones, al igual de lo que sucede en Pascal), vectores o matrices/tablas y cadenas “arrays o arreglos”; también existen otros datos especiales en lenguajes como C y C++, denominados punteros (apuntadores) y referencias.

datos estructurados	*estáticos*	*arrays* `(vectores/matrices)`
		registros `(record)`
		ficheros `(archivos)`
		conjuntos `(set)`
		cadenas `(string)`
	dinámicos	listas `(pilas/colas)`
		listas enlazadas
		árboles
		grafos

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 carácter (char), siendo los tipos lógicos, subrango y enumerativos propios de lenguajes estructurados como Pascal. Los tipos de datos compuestos están construidos basados en tipos de datos primitivos; el ejemplo más representativo es la cadena (string) de caracteres. Los tipos de datos simples pueden ser organizados en diferentes estructuras de datos: estáticas y dinámicas. Las estructuras de datos estáticas son aquellas 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: array (vectores/tablas-matrices), registros, ficheros o archivos (los conjuntos son específicos del lenguaje Pascal). Las estructuras de datos dinámicas no tienen las limitaciones o restricciones en el tamaño de memoria ocupada que son propias de las estructuras estáticas. Mediante el uso de un tipo de datos

específico, denominado puntero, es posible construir estructuras de datos dinámicas que son soportadas por la mayoría de los lenguajes que ofrecen soluciones eficaces y efectivas en la solución de problemas complejos. Las estructuras dinámicas por excelencia son las listas —enlazadas, pilas, colas—, árboles —binarios, árbol-b, búsqueda binaria— y grafos.

Arrays

Los vectores, como ya se ha comentado, pueden contener datos no numéricos, es decir, tipo "carácter"

Los vectores se almacenan en la memoria central de la computadora en un orden adyacente. Así, un vector de cincuenta números denominado NUMEROS se representa gráficamente por cincuenta posiciones de memoria sucesivas.

Cada elemento de un vector se puede procesar como si fuese una variable simple al ocupar una posición de memoria. Así,

NUMEROS[25] ← 72

almacena el valor entero o real 72 en la posición 25.ª del vector NUMEROS y la instrucción de salida

escribir (NUMEROS[25])

visualiza el valor almacenado en la posición 25.ª, en este caso 72. Esta propiedad significa que cada elemento de un vector —y posteriormente una tabla o matriz— es accesible directamente y es una de las ventajas más importantes de usar un vector: almacenar un conjunto de datos.

Registros

Un registro en Pascal es similar a una estructura en C y aunque en otros lenguajes como C# y C++ las clases pueden actuar como estructuras, en este capítulo restringiremos su definición al puro registro contenedor de diferentes tipos de datos. Un registro se declara con la palabra reservada estructura (struct, en inglés) o registro y se declara utilizando los mismos pasos necesarios para utilizar cualquier variable. Primero, se debe declarar el registro y a continuación se asignan valores a los miembros o elementos individuales del registro o estructura.

Sintaxis

estructura: nombre_clase

tipo_1: campo1

tipo_2: campo2

...

fin_estructura

registro: nombre_tipo

tipo_1: campo1

tipo_2: campo2

...

fin_registro

Los registros (estructuras) y los arrays son tipos de datos estructurados. La diferencia entre estos dos tipos de estructuras de datos son los tipos de elementos que ellos contienen. Un array es una estructura de datos homogénea, que significa que cada uno de sus componentes deben ser del mismo tipo. Un registro es una estructura de datos heterogénea, que significa que cada uno de sus componentes pueden ser de tipos de datos diferentes. Por consiguiente, un array de registros es una estructura de datos cuyos elementos son de los mismos tipos heterogéneos.

Archivos (ficheros)

Un fichero (archivo) de datos —o simplemente un archivo— es una colección de registros relacionados entre sí con aspectos en común y organizados para un propósito específico. Por ejemplo, un fichero de una clase escolar contiene un conjunto de registros de los estudiantes de esa clase. Otros ejemplos pueden ser el fichero de nóminas de una empresa, inventarios, stocks, etc.

Un archivo en una computadora es una estructura diseñada para contener datos. Los datos están organizados de tal modo que puedan ser recuperados fácilmente, actualizados o borrados y almacenados de nuevo en el archivo con todos los campos realizados.

PUNTEROS (Apuntadores)

El puntero es un tipo de dato especial que contiene la dirección de otra variable. Un puntero se declara utilizando

el asterisco (*) delante de un nombre de variable.

float *longitudOnda; /* puntero a datos float */

char *indice; /* puntero a datos char */

int *p; /* puntero a datos int */

Declaración e indirección de punteros

int m; /* una variable entera un */

int *p /* un puntero p apunta a un valor entero */

p = &m; /* se asigna a p la dirección de la variable m */

El operador de indirección (*) se utiliza para acceder al valor de la dirección contenida en un puntero.

```
float *longitudOnda;

longitudOnda = &cable1;

*longitudOnda = 40.5;
```

Punteros nulos y void

Un puntero nulo apunta a ninguna parte específica; es decir, no direcciona a ningún dato válido en memoria:

```
fp = NULL; /* asigna Nulo a fp */

fp = 0; /* asigna Nulo a fp */

if (fp != NULL) /* el programa verifica si el puntero es válido */
```

Al igual que una función void que no devuelve ningún valor, un puntero void apunta a un tipo de dato no especificado.

Punteros a valores

Para utilizar un puntero que apunte a un valor se utiliza el operador de indirección (*) delante de la variable puntero

```
double *total= &x;

*total = 34750.75;
```

El operador de dirección (&) asigna el puntero a la dirección de la variable

```
double *total;

double ctotal;

total = &ctotal;
```

Punteros y arrays

Si un programa declara

```
float *punterolista;

float listafloat [50];
```

entonces a punterolista se puede asignar la dirección del principio de listafloat (el primer elemento).

```
punterolista = listafloat;
```

o bien

```
punterolista = &listafloat [0];
```

listafloat[4] es lo mismo que *(listafloat + 4)

listafloat[2] es lo mismo que *(listafloat + 2)

Punteros a punteros

Los punteros pueden apuntar a otros punteros (doble indirección)

char **lista; /* se declara un puntero a otro puntero char */

equivale a

char *lista[];

y

char ***ptr /* un puntero a un puntero a otro puntero char */

Punteros a funciones

float *ptr; /* ptr apunta a un valor float */

float *mifun(void); /* la función mifunc devuelve un puntero a un valor float */

La declaración

int (*ptrafuncion)(void);

crea un puntero a una función que devuelve un entero, y la declaración

float (*ptrafuncion1)(int x, int y);

declara ptrafuncion1 como un puntero a una función que devuelve un valor float y requiere dos argumentos enteros.

Bibliografía

Joyanes. L Fundamentos de Programación. Algoritmos y Estructura de datos. McGraw-Hill. Mexico. 1990.