Variables tipos de datos principios básicos

Tema 3

José Ramón Paramá Gabía

José Ramón Paramá Gabía

Introducción

- Una *variable* es una abstracción de una celda o colección de celdas de la memoria del ordenador.
- Cada variable tiene unos *atributos* entre los cuales está el *tipo de dato*. Esto condiciona las operaciones que se pueden hacer sobre la variable y cómo se decodifican los valores binarios en las celdas asociadas a la variable (su valor).

Variables
atributos

José Ramón Paramá Gabía

- *Nombre:* o identificador, permite referirse a ella en el programa. En C++ se diferencian mayúsculas de minúsculas.
- *Dirección:* es la dirección en memoria de la primera celda que ocupa dicha variable.
- *Valor:* contenido de la celda o celdas de memoria asociada a la variable.
- *Tipo de dato:* condiciona el conjunto de valores que puede tomar la variable y las operaciones que se pueden hacer sobre ella

José Ramón Paramá Gabía

Variables

- En C++ y otros muchos lenguajes antes de usar una variable hay que *declararla*. Esto implica dar un nombre y un tipo de datos.
- En C++ y otros lenguajes se puede dar un valor inicial a una variable en el momento de su declaración.
- En C++ y otros lenguajes se puede declarar variables en cualquier punto del programa.

José Ramón Paramá Gabía

Constantes

- Las constantes son un tipo especial de variables que al declararlas se les tiene que dar un valor que no puede ser modificado.
- Ejemplo en C++:
  - `const double pi= 3.14159265358979323 85;`

José Ramón Paramá Gabía

Ámbito

- El *ámbito* de una variable es la parte del programa en la cual la variable existe.
- Comienza donde se declara y está relacionado con los bloques de código.
- Un *bloque de código* es una secuencia de declaraciones y sentencias. Los lenguajes proporcionan marcadores sintácticos para delimitar los bloques de código. En el caso de C++ son llaves ({}).
- Las variables que definen para un bloque de código se llaman *variable locales,* y sólo existen desde donde fueron declaradas hasta final del bloque.

José Ramón Paramá Gabía

Ámbito

- Algunos lenguajes permiten declarar variables dentro de un bloque cuyo ámbito no se restringe a ese bloque, un ejemplo son las *variables en memoria dinámica* de los lenguajes C, C++ y Java.
- Normalmente también se puede declarar variables fuera de bloques de código. Estas variables se denominan *variables globales*. Su ámbito se extiende normalmente hasta el final de un fichero fuente, aunque los lenguajes proporcionan métodos para que se pueda extender el ámbito a otros ficheros fuente.

José Ramón Paramá Gabía

Visibilidad

- Se dice que una variable es *visible* en cierta parte del programa si es posible referenciarla (usarla) en esa parte del programa.
- Dado que es posible dar el mismo nombre a distintas variables en distintos bloques de código y dado que los bloques se pueden anidar la parte de programa en que una variable es visible en ocasiones es más reducida que su ámbito.
- Se dice que una variable *oculta* a otra cuando se declaran dos variables con el mismo nombre de modo que el ámbito de las variables se solapa. En esta situación, sólo es posible referenciar una de ellas mientras que la otra permanece oculta he dicho fragmento de código.

Visibilidad ejemplo

José Ramón Paramá Gabía

```
#include <iostream>
int main(){
 int i=100;

{ int i=50;
  std::cout << "la de dentro "<<i<<std::endl;
}
 std::cout << "la de fuera "<<i<<std::endl;
 return 0;
}
```

Resultado de la ejecución

```
$ ./visible
la de dentro 50
la de fuera 100
```

Visibilidad ejemplo

José Ramón Paramá Gabía

- También es posible que en el ámbito de una variable global exista un bloque de código dentro del cual se declare una variable con el mismo nombre.
- En este caso los lenguajes suelen proporcionar un mecanismo para referenciar tanto la variable global como la local.
- En C++ este mecanismo es anteponer el operador *ámbito (::)* al nombre de la variable. Si no se antepone el operador al nombre de la variable, se entiende que se hace referencia a la variable local.

José Ramón Paramá Gabía

Tipos de datos

- El tipo de datos de una variable especifica qué *valores* puede tener y qué *operaciones* pueden realizar sobre ella.

José Ramón Paramá Gabía

Tipos de datos

- El tipo de datos condiciona el número de posiciones de memoria que son necesarias para almacenar el valor de la variable.
- El valor de una variable se almacena en memoria codificado como una o varias palabras binarias, normalmente en posiciones consecutivas.

José Ramón Paramá Gabía

Tipos primitivos

- Enteros: Normalmente hay varios tipos, que usan más o menos posiciones de memoria posibilitando representar números más o menos grandes.
- Reales: Se representan en formato de *coma flotante*. Es decir como un número (coeficiente) multiplicado por 10 elevado a un exponente entero. El coeficiente está formado por un único dígito entero, seguido de la coma decimal y un número fijo de dígitos decimales. Ej: 2,3462e-3
  - Como en el caso de los enteros suele haber varios tipos con capacidad para almacenar números más o menos mayores.

José Ramón Paramá Gabía

Tipos primitivos

- Booleano: En C no existen, un valor numérico 0 es considerado *falso*, y algo distinto de 0 *cierto*, pero otros lenguajes como C++ sí los tienen.
- Carácter: Cara carácter se representa internamente como un número, la equivalencia entre números y caracteres viene dada por un estándar, el más común es el ASCII, que en su versión extendida permite representar 256 caracteres (2^8=1 byte).

  Pero 256 caracteres no son suficientes para todos los lenguajes y actualmente se está popularizando el estándar *Unicode* que usa 16 bits (2 bytes) por carácter.

José Ramón Paramá Gabía

Tipos definidos por el programador

- Asignar un nuevo nombre a un tipo ya existente:

```
typedef int ArrayInt10[10];
```

  - Esto sería una variable de este tipo: `ArrayInt10 v;`
- Tipos enumerados: Están formados por una serie de constantes literales, que especifica el programador durante su definición.
- Estructura o record: Una estructura es una colección de variables (de posiblemente tipos diferentes), agrupadas bajo un nombre común.

```
struct Punto{
  double x;
  double y;
};
```

José Ramón Paramá Gabía

Tipos definidos por el programador

- Uniones: Una unión permite que en determinado espacio de almacenamiento en memoria se pueda almacenar valores de varios tipos. E un instante dado, sólo se puede almacenar uno de los posibles tipos de datos.

```
union Indentif{
  int i;
  char nombre;
};
```

José Ramón Paramá Gabía

Arrays

- Un array es un grupo de variables del mismo tipo, al que se hace referencia por un nombre común.
- Pueden ser de una o varias dimensiones.
- Si es de una dimensión es en esencia una secuencia de variables del mismo tipo.
- Para declararlo hay que definir el tipo de datos y el tamaño (número de variables o elementos que puede contener).
- Cada elemento se referencia por el nombre del array y un índice (o varios si el array tiene varias dimensiones) (en C++ entre corchetes), que es la posición del elemento dentro del array.

José Ramón Paramá Gabía

Arrays

- El límite inferior del rango de posibles índices es implícito en algunos lenguajes. En C++ es siempre 0.
- Como cualquier variable se puede dar valor a los elementos del array cuando se define. En ese caso en, C++ es posible omitir el tamaño del array, que el compilador obtiene del número de valores que se le dan para inicializar el array, por ejemplo:

```
double nums[]={1, 1.3, 3.4, 2, 4.5}
```

- Los elementos del array se guardan en posiciones consecutivas de memoria, en el caso de bidimensionales, por filas o por columnas.

José Ramón Paramá Gabía

Arrays

- En C++ la declaración de un array:
  - `Nombre_tipo_dato nombre_array[size_1]...[size_n];`
  - Por ejemplo: `int matriz[2][3]` consta de los elementos:

```
matriz[0][0]   matriz[0][1]   matriz[0][2]
matriz[1][0]   matriz[1][1]   matriz[1][2]
```

  - En C++ un error común acceder a posiciones de un array que no se reservaron. Por ejemplo:
  - `int a[6]` define un array de 6 posiciones pero sus índice son del 0 al 5. Así que si intento acceder a la posición a[6] estamos accediendo a una posición que no es de ese array.

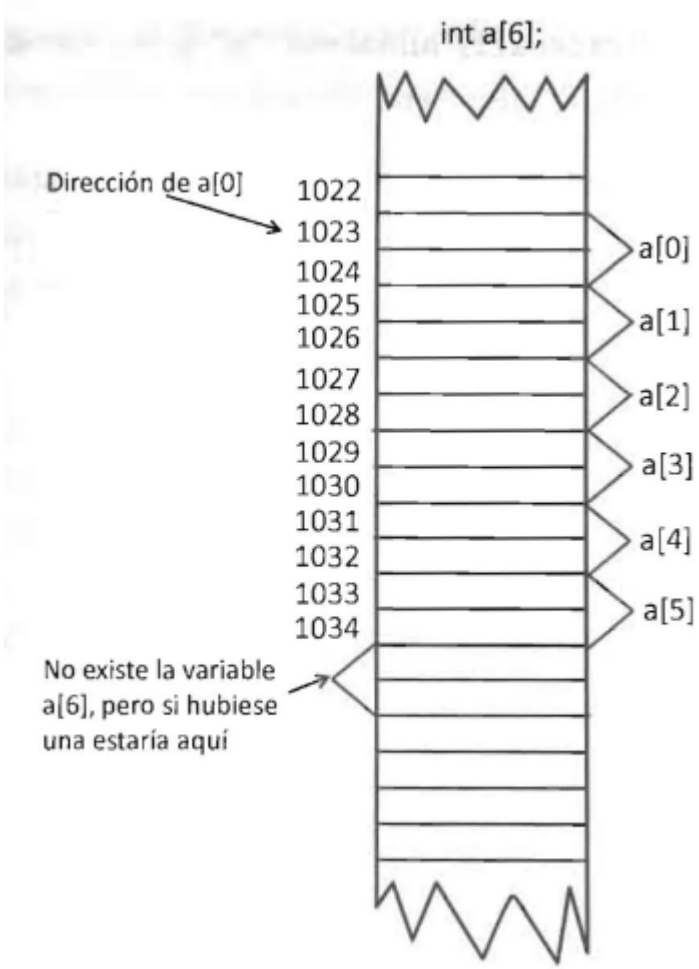
José Ramón Paramá Gabía

Array

- En C++ no se comprueba si estamos accediendo fuera del rango de valores de un array, y directamente accede a la dirección.
- Si hablamos de enteros (int) en las máquinas actuales suelen ser 4 bytes (aunque como indica el libro pueden ser 2), a[0] tiene una dirección en memoria, el compilador le suma a esa dirección 6*2=12 bytes, y accede a los siguientes 2 bytes, que pueden ser de otra variable distinta, con lo que podríamos cambiar el valor de otra variable, o no estar inicializado, lo que podría dar un error en tiempo de ejecución.

José Ramón Paramá Gabía

Array



(C. Martín, A. Urquía, M. A. Rubio. *Lenguajes de programación,* Ed. Uned, 2011)

José Ramón Paramá Gabía

Cadenas de caracteres

- Hay varios modos de representar una cadena de caracteres en C++:
  - Al estilo C introduce los caracteres en un array unidimensional, con un carácter en cada elemento y terminado en carácter nulo '\0', que señala el fin de la cadena:
    - `char cadena [] =“Hola mundo”;`

0	1	2	3	4	5	6	7	8	9	10
'H'	'o'	'l'	'a'	' '	'm'	'u'	'n'	'd'	'o'	'\0'

  - Sin embargo, C++ proporciona métodos más evolucionados para tratar con las cadenas de caracteres.

José Ramón Paramá Gabía

Punteros

- Algunos lenguajes permiten obtener las direcciones en memoria de las variables.
- También permiten acceder a la variable (para leer o modificar) proporcionando la dirección de memoria.
- Estas operaciones se realizan empleando:
  - Un tipo especial de variable denominado *puntero* que es una variable que almacena la dirección en memoria de una variable.

    El puntero también puede almacenar un valor especial; el *valor nulo*, que indica que no se apunta a ningún lado.
  - El operador *direccion-de* que aplicado a una variable nos devuelve su dirección.
  - El operador unario *indirección* que aplicado a una dirección nos devuelve el valor de la variable allí almacenada.

José Ramón Paramá Gabía

Punteros

Posición memoria	0	1	2	3	4
Valor	2	67	45	14	89

Indirección	Dirección de
*p→45	&i→2

Variable *p* puntero a *i*

Variable *i*

Programa	Salida

```
#include <iostream>
int main(){
int i=45;
int *p;
p=&i;
std::cout<<"Valor variable i: "<<i<<std::endl;
std::cout<<"Valor dirección i: "<<&i<<std::endl;
std::cout<<"Valor variable p: "<<p<<std::endl;
std::cout<<"Valor variable apuntada por p:
"<<*p<<std::endl;
return 0;
}
```

```
Valor variable i: 45
Valor dirección i: 0xbfcff26c
Valor variable p: 0xbfcff26c
Valor variable apuntada por p: 45
```

José Ramón Paramá Gabía

Variables en memoria dinámica

- Algunos lenguajes de programación (entre ellos C++) permiten declarar variables en memoria dinámica:
  - A diferencia de las variables locales, que dejan de existir cuando la ejecución del bloque donde fueron declaradas termina, las variables en memoria dinámica no dejan de existir. Se dice por ello que tienen un *tiempo de vida dinámico*.

    Las variables en memoria dinámica existen hasta que el programa termina o hasta que son eliminadas (por el programador o por el propio entorno de ejecución.
  - Una variable en memoria dinámica no tienen nombre asociado, y su declaración devuelve la dirección en memoria, así que esa dirección se debe almacenar en un puntero para poder acceder a ella.

José Ramón Paramá Gabía

Variables en memoria dinámica

- Son propensas a estos dos errores:
  - Punteros a variables eliminadas: acceder a una variable eliminada mediante un puntero.
  - Variables dinámicas perdidas: Cuando no existe ningún puntero a la variable dinámica, porque el puntero que inicialmente le apuntaba se cambió a otra dirección. En este caso ya no se puede eliminar la variable dinámica porque no tenemos modo de acceder a ella. Java no tiene este problema porque tiene un *recolector de basura,* que periódicamente busca estas variables en memoria dinámica perdidas y las elimina.