ugr Universidad de Granada

Programación de Ordenadores

Ingeniería Química

3

David Pelta

Depto de Ciencias de la Computación e I.A.

Universidad de Granada

Indice

Tipos de Datos

Entendemos por "datos" a los diferentes objetos de información con los que un programa trabaja.

Todos los datos tienen un ***tipo asociado*** con ellos que nos servirá para poder conocer con que información trabajaremos.

Ejemplos:

- El DNI de una persona es un *número entero*
- El Ph de una solución es un número con decimales
- El título de un libro es una lista de letras

Tipos de datos usuales: numéricos, caracteres, lógicos.

Tipos de Datos

La asignación de tipos a los datos tiene dos objetivos principales:

- Detectar errores de operaciones aritméticas en los programas
- Determinar como ejecutar las operaciones

Todos los objetos de un programa (variables, constantes, funciones, etc) deben tener un nombre o ***identificador.***

Tipos de Datos

Para asignar un ***identificador*** deben respetarse las siguientes reglas:

- *Serie de caracteres (letras, dígitos, y subrayado)*
- *No pueden comenzar por un número, ni contener acentos, tampoco los espacios en blanco, la eñe, las barras / y *
- *Sensibles a mayúsculas y minúsculas (Alto ≠ alto)*

Variables

- Es equivalente a una "variable" en el contexto matemático.
- Al ser un objeto de un programa, debe tener un nombre (identificador) y un tipo.
- "Fisicamente" es una porción de memoria donde se puede almacenar un dato compatible con su tipo.
- Las variables tienen que declararse antes de su uso siguiendo el esquema:

tipo nombre_de_variable

- **Ejemplos:**

```
int velocidad;
double Ph;
char inicial;
```

Tipos de Datos

Tipos de datos más importantes

Display 1.2 **Simple Types**

TYPE NAME	MEMORY USED	SIZE RANGE	PRECISION
`short` (also called `short int`)	2 bytes	−32,767 to 32,767	Not applicable
`int`	4 bytes	−2,147,483,647 to 2,147,483,647	Not applicable
`long` (also called `long int`)	4 bytes	−2,147,483,647 to 2,147,483,647	Not applicable
`float`	4 bytes	approximately 10^{-38} to 10^{38}	7 digits
`double`	8 bytes	approximately 10^{-308} to 10^{308}	15 digits
`long double`	10 bytes	approximately 10^{-4932} to 10^{4932}	19 digits
`char`	1 byte	All ASCII characters (Can also be used as an integer type, although we do not recommend doing so.)	Not applicable
`bool`	1 byte	`true`, `false`	Not applicable

The values listed here are only sample values to give you a general idea of how the types differ. The values for any of these entries may be different on your system. *Precision* refers to the number of meaningful digits, including digits in front of the decimal point. The ranges for the types `float`, `double`, and `long double` are the ranges for positive numbers. Negative numbers have a similar range, but with a negative sign in front of each number.

Variables: dar valores

Antes de usar una variable, es necesario darles algún valor. Es decir ***"inicializarla".***

La forma más directa es mediante una sentencia de asignación. Supongamos que hemos declarado:

int nro;

Ahora podemos hacer una asignación:

nro = 45;

LADO IZQUIERDO: una variable

Operador de Asignación

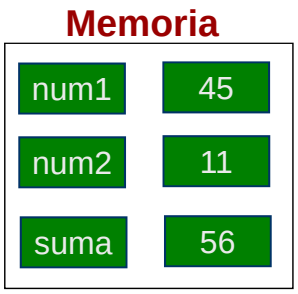
LADO DERECHO: una variable, un literal o una expresión compleja. Finaliza con punto y coma.

Variables: dar valores

Cuando se ejecuta una operación de asignación, primero se evalúa la expresión del lado derecho y luego se almacena el valor resultante en la variable indicada en el lado izquierdo.

```
int num1, num2, suma;
num1 = 45;
num2 = 11;
suma = num1 + num2;

suma =  45  +  11;
suma = 56;
```



Regla de Asignación

- Una variable en el lado derecho de una sentencia de asignación debe tener un valor antes de que la sentencia de asignación se ejecute. Hasta que un programa le da un valor a una variable, esa variable no tiene valor.

Ejemplo:

```
int x,y;
y = x + 1;
```

ERROR LÓGICO: *la variable **x** no tiene ningún valor. El valor que toma la variable **y** es impredecible!!!*

- En la izquierda de una sentencia de asignación solo pueden existir variables. La siguiente expresión no es valida:

```
int Valor_Neto, Tasas;

        Valor_Neto - Tasas =  34015;
```

Regla de Asignación

La operación de asignación es una operación destructiva: el valor almacenado en una variable se pierde o se destruye y se sustituye por el nuevo valor en la sentencia de asignación.

Literales

Un literal es una especificación de un valor concreto de un tipo de dato. Se utilizan en el lado derecho de una asignación.

Ejemplos

Valor	Tipo Literal
2	entero
45.2	real
`c'	caracter: se escribe entre ‘ ’
“Hola”	cadena de caracteres: se escribe entre " "
true	booleano
3.49e4	equivale a 3.49 x 10^{4} (34900.0)
5.89e-6	equivale a 5.89 x 10^{-6} (0.00000589)

Constantes

Son datos cuyo contenido no varía a lo largo de la ejecución del programa. Es un nombre que se da a un valor literal.

```
const double PI=3.1415;
const int SALARIO_BASE=1000;
const bool VERDAD=true;
```

Suelen usarse identificadores (nombres) sólo con mayúsculas para diferenciarlos de las variables

Ventajas de la declaración de constantes:

- *Proporcionan información*
- *Imposibilidad de cambiarlo por error (PI)*
- *Posibilidad de cambios futuros (SALARIO_BASE)*

Primer Programa en C++

```
1 // mi primer programa
2 // en C++
3 #include <iostream>
4
5 int main()
6 {
7    cout << "Que bonito es
8
9    return 0;
10    }
```

```
Que bonito es C++!
```

Comentarios

Se escriben entre **/*** y ***/** o luego de **//**.

Mejoran la legibilidad del programa y son ignorados por el compilador.

directiva del preprocesador

#include <iostream> indica al preprocesador que incluya el contenido del fichero **<iostream>**, donde se definen las o

Los programas en C++ contienen una o mas funciones, una de las cuales debe ser **main**

int indica que la función **main** "devuelve" un valor entero

return es la forma de

return 0, en este cas programa finalizo corre

Imprime el *string* de caracteres entre comillas.
Toda esta línea, incluyendo el *punto y coma* (**;**), se denomina *sentencia.*
Las sentencias terminan con *punto y coma* (**;**)

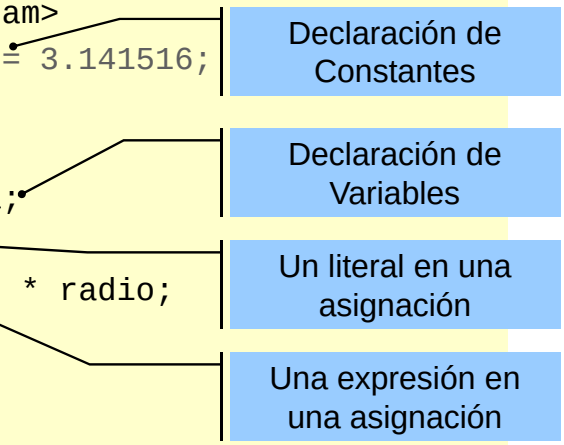
cada

Variables, Literales y Constantes

```
// Calculo del area de una circunferencia
#include <iostream>
const double PI = 3.141516;

int main(){
double radio,rta;
radio = 8.73;
rta = PI * radio * radio;

cout << "El valor del area es:" << rta;
return(0);}
```



Sugerencia

Los identificadores deben dar pistas sobre el significado o uso del objeto que están nombrando.

Aunque las siguientes expresiones sean equivalentes:

```
x = y * z;
distancia = velocidad * tiempo;
```

Esta última es más fácil de entender.

Atención

Existen "palabras reservadas" que no se pueden utilizar como identificadores. (dependen de cada lenguaje)

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			
asm	bool	catch	class	const_cast
delete	dynamic_cast	explicit	false	friend
inline	mutable	namespace	new	operator
private	protected	public	reinterpret_cast	
static_cast	template	this	throw	true
try	typeid	typename	Using	virtual
wchar_t				

Compatibilidad de Tipos

Hay que asignar a las variables valores de su mismo tipo.

Generalmente, los valores de tipo *int* pueden almacenarse en variables de tipo *double.*

```
int unEntero;
double unReal;
unEntero = 18;
unReal = unEntero;
unReal = 98;
```

unReal vale 18.0

unReal vale 98.0

La asignacion de valores de tipo *double* a variables de tipo *int* provoca la pérdida de la parte decimal.

```
int unEntero;
unEntero = 2.456;
```

unEntero vale 2

Operadores aritméticos

Se pueden utilizar con valores de tipo *int*, de tipo *double* o uno de cada tipo.

- *Si ambos son int, el resultado tendrá tipo int*
- *Si uno de ellos es double, el resultado tendrá tipo double*

Operadores

- + Suma
- - Resta
- * Multiplicación
- / Division
- % Modulo

Operadores aritméticos

División

- La división entre operandos enteros descarta la parte decimal del resultado:
  - 10 / 3 → 3 (no 3.33333)
  - 5 / 2 → 2 (no 2.5)
- La división usando al menos un operando de tipo double se comporta de la manera esperada.
  - 7.0 / 5 → 1.4
  - 7 / 5.0 → 1.4
  - 7.0 / 5.0 → 1.4

Módulo. Resto de la división entera

- 7 % 5 → 2

Ejercicios

1) Cuales de los siguientes son nombres válidos de variables?

x_1 x1 12342_hh %valor prog.cpp

2) ¿ Que hace el siguiente ejemplo?

```
int valor;
valor = 0;
valor = valor + 1;
imprimir valor;
```

3) Convierta las siguientes fórmulas a expresiones en C++

3x 3x+y $\frac{x+y}{7}$ $\frac{x+y}{z+2}$

4) ¿Cuál es la salida del siguiente programa?

```
char a,b,c;
a='b';
b='c';
c = a;
imprimir a,b,c,'c';
```

Operadores lógicos

- Son operadores binarios
- Se aplican sobre datos (y expresiones) de tipo bool:
  - AND (&&)
  - OR (||)
  - NOT (!) (este es un operador unario)
- Devuelven true o false

AND	True	False
True	True	False
False	False	False

OR	True	False
True	True	True
False	True	False

NOT	
True	False
False	True

Operadores relacionales

Son los operadores habituales de comparación de números. El resultado es de tipo bool.

(4 < 5) ⇒ true
(4 > 5) ⇒ false
((x >= 1) && (x <= 10)) ⇒ ¿x ∈ [1, 10]?

Standard algebraic equality operator or relational operator	C++ equality or relational operator	Example of C++ condition	Meaning of C++ condition
Relational operators			
>	>	`x > y`	`x` is greater than `y`
<	<	`x < y`	`x` is less than `y`
≥	>=	`x >= y`	`x` is greater than or equal to `y`
≤	<=	`x <= y`	`x` is less than or equal to `y`
Equality operators			
=	==	`x == y`	`x` is equal to `y`
≠	!=	`x != y`	`x` is not equal to `y`

Ejercicio

Dadas las variables *count = 0* y *limit = 10*, calcule el valor de las siguientes expresiones booleanas

```
(count == 0) && (limit < 20)
(limit > 20) || (count < 5)
!(count == 12)
(count == 1) && (x < y)
!(((count < 10) || ( x < y )) && (count >= 0))
```

Funciones matemáticas predefinidas

- Llevan a cabo cálculos matemáticos y devuelven un valor
- La forma para invocarlos es:

NombreFun(argumento1, argumento2, …)

Ejemplo

- *sqrt(900);*
- *cos(45)*

Los argumentos pueden ser:

- *Literales: sqrt(4)*
- *Variables: sqrt(x)*
- *Expresiones: sqrt(sqrt(x)) ó sqrt(3 - 6x)*

Algunas Funciones Matemáticas

Method	Description	Example
`ceil( x )`	rounds *x* to the smallest integer not less than *x*	`ceil( 9.2 )` is `10.0` `ceil( -9.8 )` is `-9.0`
`cos( x )`	trigonometric cosine of *x* (*x* in radians)	`cos( 0.0 )` is `1.0`
`exp( x )`	exponential function *ex*	`exp( 1.0 )` is `2.71828` `exp( 2.0 )` is `7.38906`

`fabs( x )`	absolute value of *x*	`fabs( 5.1 )` is `5.1`
		`fabs( 0.0 )` is `0.0`
		`fabs( -8.76 )` is `8.76`

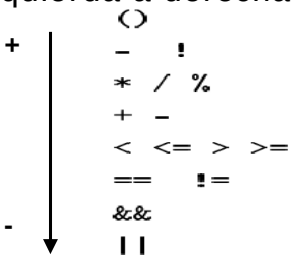
`floor( x )`	rounds *x* to the largest integer not greater than *x*	`floor( 9.2 )` is `9.0` `floor( -9.8 )` is `-10.0`
`fmod( x, y )`	remainder of *x*/*y* as a floating-point number	`fmod( 13.657, 2.333 )` is `1.992`
`log( x )`	natural logarithm of *x* (base *e*)	`log( 2.718282 )` is `1.0` `log( 7.389056 )` is `2.0`
`log10( x )`	logarithm of *x* (base 10)	`log10( 10.0 )` is `1.0` `log10( 100.0 )` is `2.0`
`pow( x, y )`	*x* raised to power *y* (*xy*)	`pow( 2, 7 )` is `128` `pow( 9, .5 )` is `3`
`sin( x )`	trigonometric sine of *x* (*x* in radians)	`sin( 0.0 )` is `0`
`sqrt( x )`	square root of *x*	`sqrt( 900.0 )` is `30.0` `sqrt( 9.0 )` is `3.0`
`tan( x )`	trigonometric tangent of *x* (*x* in radians)	`tan( 0.0 )` is `0`

Fig. 3.2 Math library functions.

Reglas de precedencia

Determinan el orden con el que se evaluan los operadores

- Paréntesis
- Funciones matemáticas
- * / %
- + -
- A igual precedencia, evaluar de izquierda a derecha
- Incluyendo todos los operadores:



Ejemplo

1. `y = 2 * 5 / 5 + 3 * 5 + 7`
2. `y = 10 / 5 + 3 * 5 + 7`
3. `y = 2 + 3 * 5 + 7`
4. `y = 2 + 15 + 7`
5. `y = 17 + 7`
6. `y = 24`

Esta forma de escribir expresiones es poco clara. Conviene usar paréntesis.

Salida de Datos

- Los valores de las variables asi como cualquier texto, se puede mostrar por pantalla a través de la instrucción *cout*
- Su sintaxis más elemental es:

```
cout << "Mensaje" << expresión1 << ....
```

- El compilador detecta el tipo de dato de las expresiones y las imprime de forma adecuada.

```
cout << "precio: " << costo_base * cant;
cout << "un calculo" << 166.386 * 92.3;
```

- Existen *secuencias de escape* que permiten dar un formato más estilizado a la salida.

Salida de Datos: formateo

Display 1.3 **Some Escape Sequences**

SEQUENCE	MEANING
\n	New line
\r	Carriage return (Positions the cursor at the start of the current line. You are not likely to use this very much.)
\t	(Horizontal) Tab (Advances the cursor to the next tab stop.)
\a	Alert (Sounds the alert noise, typically a bell.)
\\	Backslash (Allows you to place a backslash in a quoted expression.)
\'	Single quote (Mostly used to place a single quote inside single quotes.)
\"	Double quote (Mostly used to place a double quote inside a quoted string.)
The following are not as commonly used, but we include them for completeness:	
\v	Vertical tab
\b	Backspace
\f	Form feed
\?	Question mark

Entrada de Datos

- La instrucción *cin* permite ingresar datos desde el teclado.
- El valor leído debe almacenarse en una variable.
- Su sintaxis es similar a *cout* pero cambian los signos:

```
cin >> miVariable;
```

- Espera a que el usuario introduzca un valor entero desde el teclado y, cuando se pulsa la tecla Intro, lo almacena en *miVariable*
- Automáticamente se imprime el valor de la variable y además un retorno de carro
- Debe utilizarse junto con cout:

```
cout << "Introducir un número: ";
cin >> miVariable;
```

Ejemplo

```
// Prg que lee dos nros enteros por teclado
// y muestra por pantalla su suma
#include <iostream>

int main(){
  int variable1, variable2;
  int suma;

  cout << "Introducir primer dato: ";
  cin >> variable1;
  cout << "Introducir segundo dato: ";
  cin >> variable2;
  suma = variable1 + variable2;
  cout << "La suma es: " << suma << endl;
  return(0);
}
```

```
Introducir primer dato: 45
Introducir segundo dato: 72
La suma es: 117
```

Ejecutar

Ejercicios

- Supongamos que un producto cuesta 17345 euros. A cuantas pesetas equivale?. Implemente un programa que permita contestar a esta pregunta. (Recuerde 1 euro = 166.386 pesetas)
- Como debería hacer para implementar un programa que permita hacer la transformación de cualquier cantidad de euros ?
- Implemente un programa que lea un valor entero, y muestre el doble y el triple de su valor
- Escriba un programa que dados dos puntos p1 = (x1,y1) y p2=(x2,y2), calcule la distancia euclidea entre ambos

Consejos Prácticos

- Escribir programa simples y claros
- Leer los manuales de la versión de C++ que se utilice
- Comenzar los programas con un comentario que describa su propósito
- Colocar un espacio después de cada coma (,)
- Asignar nombres significativos a las variables
- Colocar una línea en blanco entre la declaración de las variables y las instrucciones
- Colocar espacios en cada lado de un operador binario
- No incluir más de una instrucción por línea
- Terminar cada programa con \n o endl

Errores Comunes

- No inicializar las variables
- Usar datos enteros y aplicar una división real
- Cadenas de desigualdades (x<z<y)
- Olvidar incluir *iostream* en un programa que usa cin ó cout
- Omitir el punto y coma al final de una instrucción
- Usar el módulo (%) sobre operandos no enteros
- Dejar espacios en blanco entre los operadores ==, !=, <= y >=
- Confundir el operador de igualdad == con el de asignación =
- Partir los identificadores con espacios en blanco *(ma in)*