

Práctica 2. TIPOS DE DATOS SIMPLES

IDENTIFICADORES

OBJETIVOS

- ✓ Conocer los distintos tipos de datos simples y su correcta utilización en la construcción de algoritmos
- ✓ Formular expresiones de forma correcta
- ✓ Evaluar distintos tipos de expresiones
- ✓ Utilizar correctamente las reglas para la construcción de identificadores

1. Tipos de datos simples

Se denomina **dato** a la expresión general que describe los objetos con los cuales opera el programa. Los tipos de datos simples son:

- **Predefinidos**
 - **Numérico**
 - **Entero:** subconjunto de los números enteros cuyo rango o tamaño dependen del lenguaje, computador utilizado y sistema operativo.
 - **Real:** subconjunto de los números reales limitado no sólo en cuanto al tamaño, sino también en cuanto a la precisión.
 - **Carácter**
 - **Alfabético:** caracteres alfabéticos, tanto mayúsculas como minúsculas.
 - **Dígito:** caracteres dígitos.
 - **Especial:** resto de caracteres de los que dispone cualquier computador.
 - **Lógico o booleano:** conjunto formado por los valores FALSO y CIERTO.
- **Definidos por el programador**
 - Subrango¹: definido a partir de un tipo ordinal y finito, especificando dos constantes de ese tipo, que actúan como límite inferior y superior del conjunto de datos de ese tipo
 - Enumerado: compuesto de un conjunto de valores referenciados por identificadores

¹ En el lenguaje de programación C no es posible definir el tipo de dato subrango

Ejercicio Resuelto 1. ¿Qué tipos de datos simples emplearías para almacenar la siguiente información?:

		Tipos posibles en C
a)	Edad:	Numérico Entero
b)	D.N.I.:	Numérico Entero
c)	Altura:	Numérico Entero o Numérico Real
d)	Sexo:	Carácter o Enumerado
e)	¿Casado?:	Carácter, Lógico o Enumerado
f)	I.V.A. a aplicar:	Numérico real

Tipos posibles en C

unsigned, short, int

unsigned, int

unsigned, int, float, double

char, enum T_Sexo {hombre, mujer}

char, boolean enum T_Casado {casado, soltero}

float, double (si queremos guardar 0.18)

2. Operadores aritméticos

Operación	Operador algorítmico
Suma	+
Resta	-
Multiplicación	*
División	/
Resto de la división	%
Potencia	pow (base, exponente)

¡Cuidado con la división!

Según los tipos de datos con los que apliques dicho operador, obtendrás un resultado diferente. Por ejemplo, si calculas $5/2$ obtendrás como resultado 2, en cambio, si calculas $5/2.0$, obtendrás como resultado 2.5.

Potencias

Si necesitas calcular 4^8 , tendrás que escribir: `pow(4, 8)` y asignar a una variable el resultado `var=pow(4, 8);`

Debes incluir la librería `cmath` para poder usar la función `pow()`, esto es, nada más empezar el código tras `#include <iostream>` deberás escribir `#include <cmath>`

Ejercicio Resuelto 2.**a) Convierte en expresiones algorítmicas las siguientes expresiones algebraicas:**

a)	$a^2 + b^2$	$a * a + b * b$
b)	$(a + b)^2$	$(a + b) * (a + b)$
c)	$\sqrt[3]{b} + 34$	$b^{(1/3)} + 34$
d)	$\sqrt[3]{b + 34}$	$(b + 34)^{(1/3)}$
e)	$\frac{c + d}{u + \frac{w}{b}}$	$(c + d) / (u + w / b)$
f)	$c + \frac{d}{u} + \frac{g}{b}$	$c + d / u + g / b$
g)	$\frac{a}{b} (z + w)$	$a / b * (z + w)$

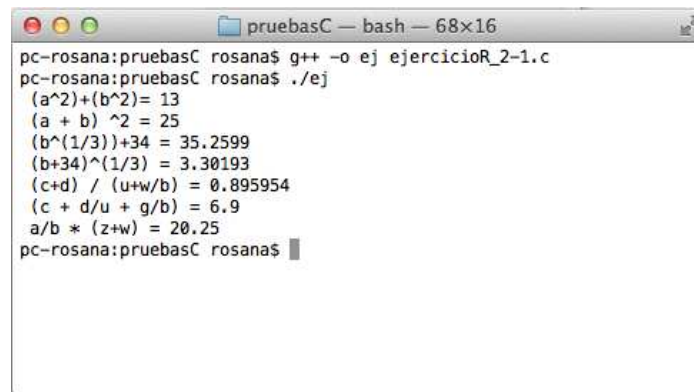
b) Comprueba mediante lenguaje C que el resultado obtenido es el mismo tanto evaluando la expresión algebraica, como ejecutando la expresión algorítmica.

Para probarlo en C podríamos compilar y ejecutar el siguiente programa **ejercicioR_2-1.c**

```

1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4
5  int main() {
6      double a, b, c, d, u, w, g, z;
7
8      a = 3.0;
9      b = 2.0;
10     c = 4.5;
11     d = 3.25;
12     u = 5.0;
13     w = 7.3;
14     g = 3.5;
15     z = 6.2;
16     cout << "(a^2)+(b^2)= " << pow(a,2)+ pow(b,2) << endl;
17     cout << "(a + b) ^2 = " << pow((a+b),2) << endl;
18     cout << " (b^(1/3))+34 = " << pow(b,(1/3.0))+34 << endl;
19     cout << " (b+34)^(1/3) = " << pow(b+34,(1/3.0)) << endl;
20     cout << " (c+d) / (u+w/b) = " << ((c+d)/(u+w/b)) << endl;
21     cout << " (c + d/u + g/b) = " << (c + d/u + g/b) << endl;
22     cout << " a/b * (z+w) = " << (a/b * (z+w)) << endl;
23 }
```

El resultado de la ejecución tras la compilación será lo siguiente:



```

pc-rosana:pruebasC rosana$ g++ -o ej ejercicioR_2-1.c
pc-rosana:pruebasC rosana$ ./ej
(a^2)+(b^2)= 13
(a + b) ^2 = 25
(b^(1/3))+34 = 35.2599
(b+34)^(1/3) = 3.30193
(c+d) / (u+w/b) = 0.895954
(c + d/u + g/b) = 6.9
a/b * (z+w) = 20.25
pc-rosana:pruebasC rosana$
```

`pow()` es una función de C incluida en la librería `cmath()` que calcula la potencia de un número. El concepto de función se verá en el tema 4 de la asignatura.

3. Operadores lógicos y relacionales

Estos operadores se emplean para comparar datos y producen como resultado un valor lógico (true/false).

Operador	Sintaxis
Menor que	<
Menor o igual que	<=
Mayor que/	>
Mayor o igual que	>=
igual que	==
distinto	!=
Negación lógica	!
AND lógico	&&
OR lógico	

Importante

Cuando queramos mostrar en pantalla el resultado de una expresión lógica, es aconsejable guardar el resultado de la expresión en una variable y después mostrar el valor de esa variable. Las expresiones lógicas darán como resultado 1 ó 0, 1 si es verdadero, 0 si es falso, por lo que la variable en la que almacenamos el resultado debe ser de tipo `int`.

Ejercicio Resuelto 3. Evalúa las siguientes expresiones:

- `a>b`
- `a>d && c<b`
- `d>a || c>=b`
- `!(a!=b)`

siendo `a = 4.0`, `b = 7.0`, `c = 8.5`, `d = 1.5`

```

1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4
5  int main() {
6      double a, b, c, d;
7      int aux;
8
9      a = 4.0;
10     b = 7.0;
11     c = 8.5;
12     d = 1.5;
13
14     aux=a>b;
15     cout << "a>b " << aux << endl;
16
17     aux = a>d && c<b;
18     cout << "a>d && c<b " << aux << endl;
19
20     aux = d>a || c>=b;
21     cout << "d>a || c>=b" << aux << endl;
22
23     aux = !(a!=b);
24     cout<< "!(a!=b) = " << aux <<endl;
25 }
```

El resultado tras la compilación y ejecución sería el siguiente:



```

pc-rosana:pruebasC rosana$ g++ -o ej ejercicioR_2-2.c
pc-rosana:pruebasC rosana$ ./ej
a>b 0
a>d && c<b 0
d>a || c>=b1
!(a!=b) = 0
pc-rosana:pruebasC rosana$
```

4. Identificadores


Identificadores son los nombres que utiliza el programador para referenciar los datos y otros elementos del programa (constantes simbólicas, variables, funciones y procedimientos u otros objetos que manipulan el algoritmo), permitiendo así su definición en una posición de la memoria del ordenador.

La regla para construir un identificador establece que:

- **Debe resultar significativo**, sugiriendo lo que representa. Es decir, el nombre asignado debe tener relación con la información que contiene, pudiéndose emplear abreviaturas que sean significativas.
- **No podrá coincidir con palabras reservadas**, propias del lenguaje de programación.
- **La longitud no debe ser excesivamente larga**. De todos modos, el número máximo de caracteres que se pueden emplear depende del compilador utilizado.
- **Comenzará siempre por un carácter alfabético** y los siguientes podrán ser letras, dígitos o el símbolo de subrayado. Sólo se permitirán los caracteres alfabéticos correspondientes a los códigos ASCII menores de 127. No debe contener espacios.
- Según el lenguaje de programación podrá ser utilizado indistintamente o no, en mayúsculas o en minúsculas.

Algunas de las nomenclaturas utilizadas por la mayoría de programadores son:

- Todo el identificador en minúsculas, separando las diferentes palabras que pueda tener con el subrayado (Ejemplo: codigo_cliente).
- Todo el identificador en mayúsculas, separando las diferentes palabras que pueda tener con el subrayado (Ejemplo: CODIGO_CLIENTE).
- Todo el identificador en minúscula excepto las iniciales de cada palabra (Ejemplo: CodigoCliente).
- Utilización de nombres completos.
- Utilización de abreviaturas. Es conveniente utilizar abreviaturas con la misma longitud. (Ejemplo: cod_cli).

 Recordad que es indispensable no cambiar arbitrariamente de nomenclatura para mantener una coherencia en todos nuestros programas.



Debes tener en cuenta que el lenguaje C distingue entre mayúsculas y minúsculas, es decir, no es lo mismo la variable `nota` que la variable `Nota`

Ejercicio Resuelto 4. En una asignatura de la titulación la práctica vale el 60% de la nota y la teoría el 40%. Aquí tienes un programa en lenguaje C para calcular la nota de un alumno

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      float teoria, practica, global;
6
7      cout << "Introduce la nota de teoría:";
8      cin >> teoria;
9      cout << "Introduce la nota de prácticas:";
10     cin >> practica;
11     global = 0.4*teoria + 0.6*practica;
12     cout << "La nota global es un " << global << endl;
13 }
```

```

pc-rosana:pruebasC rosana$ g++ -o ej ejercicioR_2-4.c
pc-rosana:pruebasC rosana$ ./ej
Introduce la nota de teoría:7
Introduce la nota de prácticas:6
La nota global es un 6,4
pc-rosana:pruebasC rosana$

```

Fíjate en que aunque el programa anterior es equivalente a éste que ponemos a continuación, es mucho más claro porque los nombres de variable son mucho más significativos

```

1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      float t,p,nl;
6
7      cout << "Introduce la nota de teoría:";
8      cin >> t;
9      cout << "Introduce la nota de prácticas:";
10     cin >> p;
11     nl = 0.4*t + 0.6*p;
12     cout << "La nota global es un " << nl << endl;
13 }

```

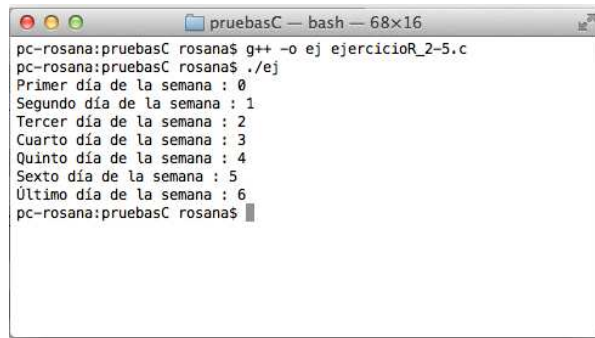
Ejercicio Resuelto 5. Escribe un programa que defina un tipo enumerado para los días de la semana, una variable del tipo enumerado e imprima los días de la semana.

```

1  #include <iostream>
2  using namespace std;
3
4
5  // Definición del tipo enumerado
6  enum DiasSemana {lunes, martes, miercoles, jueves, viernes, sabado,
7  domingo};
8
9  int main() {
10     enum DiasSemana dia; //definición de una variable de tipo DiasSemana
11
12     dia = lunes;
13
14     cout << "Primer día de la semana : " << dia << endl;
15     cout << "Segundo día de la semana : " << dia+1 << endl;
16     cout << "Tercer día de la semana : " << dia+2 << endl;
17     cout << "Cuarto día de la semana : " << dia+3 << endl;
18     cout << "Quinto día de la semana : " << dia+4 << endl;
19     cout << "Sexto día de la semana : " << dia+5 << endl;
20     cout << "Último día de la semana : " << dia+6 << endl;
21 }

```

Tras la compilación y ejecución vemos lo siguiente:



```

pc-rosana:pruebasC rosana$ g++ -o ej ejercicioR_2-5.c
pc-rosana:pruebasC rosana$ ./ej
Primer día de la semana : 0
Segundo día de la semana : 1
Tercer día de la semana : 2
Cuarto día de la semana : 3
Quinto día de la semana : 4
Sexto día de la semana : 5
Último día de la semana : 6
pc-rosana:pruebasC rosana$

```

Importante

Es interesante resaltar que no hay formato de impresión para tipos enumerados. Se proyectan sobre sus valores enteros. Cada uno de los elementos de la enumeración lleva asociado un valor entero. Por defecto, el primer elemento tiene asociado el valor 0, el segundo el valor 1 y así sucesivamente. Debido a que un valor de tipo enumerado se considera como un entero, se pueden utilizar los operadores aritméticos como ocurre en la expresión `dia=dia+1`.

PRUEBA Y DEPURACIÓN

En inglés, a un error de programa se le conoce como **bug** (insecto), y al proceso de eliminación de errores se le llama **debugging**, es decir, depuración. Son aquellos que se producen cuando el programa viola la sintaxis, es decir, las reglas de gramática del lenguaje. Se suelen detectar por el compilador durante el proceso de compilación. Errores comunes: falta punto y coma, olvido de dobles comillas al cerrar una cadena, etc.

Cuando probamos un programa:

- debemos hacerlo con la peor intención con el objetivo de encontrarle fallos,
- durante la fase de pruebas cuantos más errores encontremos mejor.

TIPOS DE ERRORES

Errores de sintaxis. Son los errores que se producen cuando el programa viola la sintaxis, es decir, las reglas de gramática del lenguaje. Se suelen detectar por el compilador durante el proceso de compilación. Errores comunes: falta punto y coma, olvido de dobles comillas al cerrar una cadena, etc.

Errores lógicos. representan errores del programador en el diseño del algoritmo y posterior programa. Los errores lógicos son más difíciles de encontrar y aislar ya que no suelen ser detectados por el compilador. Únicamente tras una comprobación de todo el programa y de los resultados previstos es posible su detección.

Por ejemplo: `res = var + var;` (para calcular el cuadrado por error escribimos una suma cuando realmente queremos escribir un signo de multiplicación; si además probamos con `res= 2+2;` no detectamos el error; debemos hacer una traza/seguimiento de todos los pasos y probar con varios valores para detectar este error lógico).

Errores de regresión. Aquellos que se crean accidentalmente cuando se intenta corregir un error lógico. Siempre que se corrige un error se debe comprobar totalmente la exactitud (corrección) para asegurarse que se elimina el error que se está tratando y no produce otro error.

MENSAJES DE ERROR

Los compiladores emiten mensajes de error o de advertencia durante las fases de compilación, de enlace o de ejecución de un programa. Se pueden agrupar en tres grandes bloques.

- **Errores fatales.** Son raros. Algunos de ellos indican un error interno del compilador.
- **Errores de sintaxis.** Son los errores típicos de sintaxis, errores de línea de órdenes y errores de acceso a memoria o disco. El compilador terminará la fase actual de compilación y se detendrá.
- **Advertencias (*warning*).** No impiden la compilación. Indican condiciones que son sospechosas, pero son legítimas como parte del lenguaje.