Tema 3: Operadores y expresiones

Introducción a la programación I

Marcos Novalbos
Elena García Gamella

Índice

1. Introducción

- Los programas se basan esencialmente en la realización de numerosas operaciones aritméticas y matemáticas de diferente complejidad.
- Cada una de ellas se divide en otro conjunto de operaciones de lenguaje máquina realizadas dentro de la CPU, concretamente en la ALU.
- Los operadores fundamentales son:
  - Aritméticos, lógicos y relacionales
  - De manipulación de bits
  - Condicionales
  - Espaciales
- Por otro lado, se realizan a menudo conversiones de tipos de datos
- Una expresión es una sucesión de operadores y operandos debidamente relacionados para formar expresiones matemáticas que determinan un cálculo.
- Existen reglas que sigue el compilador cuando concurren en la misma expresión diferentes tipos de operadores

2. Operador de asignación

- Tiene la siguiente sintaxis:

```
variable = expresión;
```

Donde variable es un identificador válido declarado como variable

- Es asociativo por la derecha, luego se pueden realizar asignaciones múltiples:

```
a = b = c = 10;
```

- Esta propiedad permite inicializar varias variables de una vez.

2. Operador de asignación

Símbolo	Uso	Descripción
=	a = b	Asigna el valor de b a a
+=	a += b (a = a + b)	Suma b y a y asigna el resultado a la variable a
-=	a -= b (a = a - b)	Resta b de a y lo asigna el resultado a la variable a
*=	a *= b (a = a * b)	Multiplica a por b y asigna el producto a la variable a
/=	a /= b (a = a / b)	Divide a por b y asigna el cociente a la variable a
%=	a %= b (a = a % b)	Divide a por b y asigna el resto a la variable a

3. Operadores aritméticos

- Son usados en operaciones básicas
- El orden de evaluación de una expresión lo determina la prioridad y asociatividad de los operadores
- Los paréntesis pueden cambiar el orden usual de evaluación de una expresión

Operador	Significado	Ejemplo
+	Suma	a + b
-	Resta	a - b
*	Multiplicación	a * b
/	Cociente División	a / 5
%	Resto División de Enteros (Operación "Módulo")	a % 5

4. Operadores de incremento y decremento

- Incremento ++
- Decremento --
- Pueden utilizarse como sufijo o prefijo.
  - Post-incremento: m = n++; se realiza primeramente la asignación y después se incrementa n en una unidad
  - Pre-incremento: m = ++n; primero se incrementa n en una unidad y luego se realiza la asignación

Incremento	Decremento
++n, n++	--n, n--
n += 1	n -= 1
n = n + 1	n = n - 1

5. Operadores relacionales

- C no tiene tipo de datos lógicos. En su lugar utiliza el tipo int, con el valor entero 0 como falso y distinto de cero para verdadero. Sintaxis:

```
expresion operador_relacional expresión
```

- Se pueden aplicar a operandos de cualquier tipo de datos (char, int, …)

Operador	Significado	Ejemplo
==	Igual a	a == b
!=	No igual a	a != b
>	Mayor que	a > b
<	Menor que	a < b
>=	Mayor o igual que	a >= b
<=	Menor o igual que	a <= b

5. Operadores relacionales

- Ejemplo 1:
  - Generar un programa que pida dos números enteros por pantalla, y que compruebe si son iguales. Almacenará el resultado en una variable entera y lo mostrará por pantalla. No se usarán operadores "if/then", sólo se almacena el resultado de la comprobación.

5. Operadores relacionales

- Ejemplo 1:
  - Generar un programa que pida dos números enteros por pantalla, y que compruebe si son iguales. Almacenará el resultado en una variable entera y lo mostrará por pantalla. No se usarán operadores "if/then", sólo se almacena el resultado de la comprobación.

```
int op1=0,op2=0;
int result=0;
printf("Introduzca dos valores enteros\n");
scanf("%d %d", &op1,&op2);
result= (op1==op2);
printf("El resultado es %d\n",result);
```

6. Operadores lógicos

- Se utilizan con expresiones para devolver un valor verdadero (cualquier entero distinto de cero) o un valor falso (0).

Símbolo	Significado	Descripción
!	not	Produce falso si su operando es verdadero y viceversa
&&	and	Produce verdadero sólo si ambos operandos son verdaderos. Si cualquiera de los operandos es falso produce falso
||	or	Produce verdadero si cualquiera de los operandos es verdadero y falso solo si ambos operandos son falsos.

- `!` mayor prioridad que `&&` mayor prioridad que `||`
- La asociatividad es de izquierda a derecha.

6. Operadores lógicos

- Tablas de verdad de los operadores lógicos.

A	B	AND (&&)	OR (||)	XOR (^)
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

6. Operadores lógicos

- Ejemplo 2:
  - Generar un programa que pida 4 números en coma flotante, y almacenarlos en 4 variables distintas. Mostrar el resultado de calcular las siguientes condiciones:
    - Si el primero es mayor que el segundo **"y"** el primero es mayor que el tercero
    - Si el cuadrado del tercero es menor que el segundo por el cuarto **"o"** el tercero por 2 menor que el segundo
    - Si el resto de la división entre el primero y el cuarto es mayor que el tercero **"y"** el primero es mayor que el cuarto
    - Si la división entre el segundo y el cuarto **"no es"** distinta al primero

6. Operadores lógicos

Ejemplo 2:

```
float op1=0,op2=0,op3=0,op4=0;
int opLogica1=0,opLogica2=0,opLogica3=0,opLogica4=0;

printf("Introduzca cuatro valores en coma fotante \n");
scanf("%f %f %f %f", &op1,&op2,&op3,&op4);

opLogica1= (op1>op2) && (op1>op3);
opLogica2= (op3*op3)<(op2*op4) || ((op3*2)<op2);
opLogica3= ((int)op1%(int)op4)>op3 && (op1>op4);
opLogica4= !((op2/op4)!=op1);

printf("El resultado de las operaciones son\n"
       "\t1: %d\n"
       "\t2: %d\n"
       "\t3: %d\n"
       "\t4: %d\n", opLogica1, opLogica2, opLogica3, opLogica4);
```

7. Operadores de manipulación de bits

- Ejecutan operaciones sobre cada uno de los bits de los operandos.
- Los operadores de manipulación de bits se aplican sólo a variables y constantes char, int y long y no a datos en coma flotante.

Operador	Uso	Significado
&		AND lógica bit a bit
|		OR lógica bit a bit
^		XOR lógica bit a bit
~		Complemento a uno (inversión de todos los bits)
<<	a << n	Desplazamiento de n posiciones a la izquierda de los bits del operando a. Equivale a multiplicar el número a por 2^n.
>>	a >> n	Desplazamiento de n posiciones a la derecha de los bits del operando a. Equivale a dividir el número a por 2n.

7. Ejemplos: operadores manipulación de bits

- Ejemplo 3:
  - Escribir un código que muestre si el tercer bit (contando desde la derecha) de un número entero de 8 bits sin signo leído por terminal, es un 1 o un 0.

7. Ejemplos: operadores manipulación de bits

- Ejemplo 3 (resultado 1):

Usando operadores de desplazamiento

```
unsigned char dato=0;
unsigned char tercerBit=0;
unsigned int esUno=0;

printf("Introduzca un número entero de 8 bits válido\n")
scanf("%c",&dato);

tercerBit= 1<< 2; /* desplazar 2 posiciones el 00000001 */

esUno = tercerBit & dato; // hacemos un "and" bit a bit

printf("El resultado del and bit a bit es:%d\n", esUno);
```

7. Ejemplos: operadores manipulación de bits

- Ejemplo 3 (resultado 2):

Usando máscaras de bit

```
#define  TERCERBIT_HEX     0x04
#define  TERCERBIT_BIN     0b00000100

unsigned char dato=0;
unsigned int esUno=0;

printf("Introduzca un número entero de 8 bits válido\n")
scanf("%c",&dato);

esUno = TERCERBIT_HEX & dato;

printf("El resultado es:%d\n", esUno);
```

7. Ejemplos: operadores manipulación de bits

Ejemplo 4:

- Crear una función que calcule el resultado de multiplicar un número por 2 elevado a un exponente, y que lo devuelva como resultado. El número y el exponente deberán pedirse al usuario mediante scanf y pasarse por parámetros a la función.El exponente debe ser mayor que cero. No se puede usar el operador "*" (multiplicar).

7. Ejemplos: operadores manipulación de bits

Ejemplo 4:

- Crear una función que calcule el resultado de multiplicar un número entero por 2 elevado a un exponente, y que lo devuelva como resultado. El número y el exponente deberán pedirse al usuario mediante scanf y pasarse por parámetros a la función. El exponente debe ser mayor que 0. No se puede usar el operador "*" (multiplicar).

```
int operacion(int num, int exponente)
{
      int res=num<<exponente;
      return res;
}
```

8. Operadores de direcciones

- Permiten manipular las direcciones de memoria de las variables y objetos en general

Operador	Significado
*	Lee o modifica el valor apuntado por la expresión. Se corresponde con un puntero y el resultado es del tipo apuntado
&	Devuelve un puntero (dirección de memoria) al objeto utilizado como operando, que debe ser una variable dotada de dirección de memoria.
.	Permite acceder a un miembro de un objeto agregado (unión, estructura)
->	Accede a un miembro de un objeto agregado (unión, estructura) por el operando de la izquierda, siendo este un puntero ó dirección de memoria.

9. Operadores especiales

Operador	Significado
, (coma)	Permite combinar dos o más expresiones separadas por comas en una misma línea. Se evalúa primero la expresión de la izquierda y luengo las restantes expresiones de izda a dcha.
( )	Es el operador que encierra los parámetros de funciones. Efectúa las conversiones explícitas de tipo
[ ]	Sirve para dimensionar los arrays y designar un elemento de un array.

10. Conversiones de tipos

- En las expresiones se pueden mezclar variables de distintos tipos, es decir, un operador binario puede tener a su izquierda un operando de un tipo, como por ejemplo int y a su derecha un operando de otro tipo, como por ejemplo double.

  En estos casos el compilador automáticamente se encargará de realizar las conversiones de tipos adecuadas. Estas conversiones trabajan siempre promoviendo el tipo “inferior” al tipo “superior”, obteniéndose un resultado que es del tipo “superior”.

- En otras ocasiones es conveniente forzar una conversión de tipos en algunas de las variables que forman parte de una expresión.

10. Conversiones de tipos

- Según lo indicado anteriormente las conversiones de tipo pueden ser:
  - **Explícitas**. Solicitadas específicamente por el programador
  - **Implícitas**. Se ejecutan automáticamente.

- **Conversiones explicitas**
  - C fuerza la conversión explícita de tipos mediante el operador **(cast).**
  - El operador cast tiene el formato:

```
(<tipo_de_dato>) <variable o expresión>

/* convierte al tipo de dato indicado la variable o expresión */
```

10. Ejemplos: Conversiones de tipos

- Ejemplo 5:
  - Convertir el numero entero 100 a carácter e imprimirlo por pantalla.

10. Ejemplos: Conversiones de tipos

- Ejemplo 5:
  - Convertir el numero entero 100 a carácter e imprimirlo por pantalla.

```
int ent = 100;
char car = (char)ent;
printf("%c", car);
```

10. Conversiones de tipos

- **Conversión Implícita**
  - C hace conversiones de tipo automáticamente cuando:
    - Se asigna un valor de un tipo a una variable de otro tipo,
    - Se combinan tipos mixtos en expresiones
    - Se pasar argumentos a funciones
  - Los tipos fundamentales (básicos) pueden ser mezclados libremente en asignaciones y expresiones.
  - Las conversiones se ejecutan automáticamente. Los operandos de tipo más bajo se convierten en los de tipo más alto de acuerdo con las siguientes reglas:
    - Si cualquier operando es de tipo char, short o enumerado se convierte en tipo int
    - Si los operandos tienen diferentes tipos, las conversiones trabajan siempre promoviendo el tipo "inferior" al tipo "superior", obteniéndose un resultado que es del tipo "superior".
    - La siguiente lista determina a qué operador se convertirá

      `int, unsigned int, long, unsigned long, float, double`
    - El orden de conversión es de izda a drcha

10. Ejemplos: Conversiones de tipos

- Ejemplo 6:
  - Crear un programa que pida un dato en formato coma flotante, y muestre el resultado de convertirlo a "integer".

10. Ejemplos: Conversiones de tipos

- Ejemplo 6:
  - Crear un programa que pida un dato en formato coma flotante, y muestre el resultado de convertirlo a integer.

```
int floatToInt(float a)
{
     return (float) a;
}
…
float a=0;
int res=0;
printf("introduzca un número en coma flotante\n");
scanf("%f",&a);
res=floatToInt(a);
printf("El resultado es %d\n", res);
```

10. Ejemplos: Conversiones de tipos

Ejemplo 7:

- Definir 4 variables a, b, c y d de tipo long, unsigned char, int y float respectivamente. Realiza la operación a + b * c / d y déjalo en la variable f. Determina el tipo de la variable f

10. Ejemplos: Conversiones de tipos

Ejemplo 7:

- Definir 4 variables a, b, c y d de tipo long, unsigned char, int y float respectivamente. Realiza la operación **a + b * c / d** y déjalo en la variable f. Determinar el tipo de la variable f

```
long          a;
unsigned char b;
int           c;
float         d;
```

  - `b * c -> b se convierte en int.  Las variables b y c son de tipo int, por tanto  se ejecuta el producto y el resultado es un int`
  - `(b *c) / d -> como d es float y b*c es un int este ultimo se convierte a float. Así el resultado de la division es un float.`
  - `Por ultimo, a se convierte a float para ejecutar la suma y el resultado ,f, es un` **`float`**`.`

10. Ejemplos: Conversiones de tipos

Ejemplo 7:

- Definir 4 variables a, b, c y d de tipo long, unsigned char, int y float respectivamente. Realiza la operación **a + b * c / d** y déjalo en la variable f. Determinar el tipo de la variable f

¿Qué pasa si se define f com int?

```
long          a;
unsigned char b;
int           c;
float         d;
```

- `b * c -> b se convierte en int. Las variables b y c son de tipo int, se ejecuta el producto y el resultado es un int.`
- `(b *c) / d -> como d es float y b*c es un int este ultimo se convierte a float. Así el resultado de la division es un float.`
- `Por ultimo, a se convierte a float para ejecutar la suma y el resultado ,f, es un` **`float`**`.`

10. Ejemplos: Conversiones de tipos

Ejemplo 7:

- Definir 4 variables a, b, c y d de tipo long, unsigned char, int y float respectivamente. Realiza la operación a + b * c / d y déjalo en la variable f. Determina el tipo de la variable f

¿Qué pasa si se define f com int?

```
long          a;
unsigned char b;
int           c;
float         d;
```

- b * c -> b se convierte en int. Las variables b y c son de tipo int, se ejecuta el producto y el resultado es un int.
- (b *c) / d -> como d es float y b*c es un int este ultimo se convierte a float. Así el resultado de la division es un float.
- Por ultimo, a se convierte a float para ejecutar la suma y el resultado ,f, es un **float**.

Respuesta: El float resultado de la operación se convierte en un entero truncando su valor

10. Ejemplos: Conversiones de tipos

Ejemplo 8:

- Crear una función que reciba una variable entera y devuelva la dirección de memoria en la que se encuentra. Desde el programa principal, almacenar la dirección de memoria devuelta. Para poder almacenar direcciones de memoria se usarán variables de tipo entero largo.

10. Ejemplos: Conversiones de tipos

Ejemplo 8:

- Crear una función que reciba una variable entera y devuelva la dirección de memoria en la que se encuentra. Desde el programa principal, almacenar la dirección de memoria devuelta. Para poder almacenar direcciones de memoria se usarán variables de tipo entero largo.

```
/* no es válida */
long int darDirMem(int var)
{
    long int dir = (long int)&var;
    return dir;
}
```

Ojo, definición no válida

10. Ejemplos: Conversiones de tipos

Ejemplo 8:

- Crear una función que reciba una variable entera y devuelva la dirección de memoria en la que se encuentra. Desde el programa principal, almacenar la dirección de memoria devuelta. Para poder almacenar direcciones de memoria se usarán variables de tipo entero largo.

Mejor con una macro

```
#define DIRMEM(a) (long int)&a
```


U-tad
CENTRO UNIVERSITARIO
DE TECNOLOGÍA Y ARTE DIGITAL