

Tema 3: Operadores y expresiones

Introducción a la programación I

Ana Isabel Sierra de las Heras
Marcos Novalbos
Francisco Javier Garcia Algarra
Rodrigo Alonso Solaguren-Beascoa
Alfonso Castro Escudero

Índice

1. Introducción
2. Operador de asignación
3. Operadores aritméticos
4. Operadores de asignación
5. Operadores de incremento y decremento
6. Operadores relacionales
7. Operadores lógicos
8. Operadores de manipulación de bits
9. Operadores de direcciones
10. Operadores especiales
11. Conversiones de tipos

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.

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")	$a \% 5$

3. Ejercicio 1. Entrada/Salida

- Este programa calcula el área de un triángulo.

3. Ejercicio 1. Entrada/Salida

- Este programa calcula el área de un triángulo.

```
#include <stdio.h>
```

```
int main() {
```

```
    float base, altura, area;
```

```
    printf("Ingrese la base del triángulo: ");
```

```
    scanf("%f", &base);
```

```
    printf("Ingrese la altura del triángulo: ");
```

```
    scanf("%f", &altura);
```

```
    area = (base * altura) / 2;
```

```
    printf("El área del triángulo es: %f\n", area);
```

```
    return 0;
```

```
}
```

¿De qué tipo se declaran la base y la altura?
¿Qué ocurre cuando base y altura son int?
¿Qué ocurre cuando base y altura son float?

3. Operadores aritméticos

- El operador aritmético `*` en C se utiliza para realizar la multiplicación de dos números. La forma en que actúa depende de los tipos de datos de los factores
 - Ambos factores son del mismo tipo
 - Si ambos factores son del mismo tipo, la multiplicación se realiza normalmente y el resultado tendrá el mismo tipo.
 - Un factor es de tipo `int` y el otro es de tipo `float` o `double`:
 - El resultado se convierte en un número en coma flotante para preservar la precisión de los decimales

3. Operadores aritméticos

- El operador aritmético / en C se utiliza para realizar la división entre dos números. La forma en que actúa depende de los tipos de datos de los operandos (numerador y denominador)
 - Ambos operandos son del mismo tipo
 - Si tanto el numerador como el denominador son del mismo tipo, la división se realiza normalmente y el resultado tendrá el mismo tipo.
 - La división de dos enteros produce un resultado entero, y cualquier parte decimal se trunca
 - El numerador es de tipo int y el denominador es de tipo float o double
 - El resultado se convierte en un número en coma flotante. Esto se hace para preservar la parte decimal del resultado
 - El numerador es de tipo float o double y el denominador es de tipo int:
 - El resultado se mantiene como un número en coma flotante para preservar la precisión de los decimales
- El operador / en C realiza la división de acuerdo con las reglas de conversión de tipos para preservar la precisión de los resultados, y el tipo del resultado depende de los tipos de datos de los operandos involucrados en la operación de división

3. Ejercicio 2. Entrada/Salida

- Este programa verifica si un número ingresado por el usuario es par o impar utilizando el operador de módulo %.

3. Ejercicio 2. Entrada/Salida

- Este programa verifica si un número ingresado por el usuario es par o impar utilizando el operador de módulo %

```
#include <stdio.h>

int main() {
    int num;

    printf("Ingrese un número entero: ");
    scanf("%d", &num);

    if (num % 2 == 0) {
        printf("%d es un número par.\n", num);
    } else {
        printf("%d es un número impar.\n", num);
    }

    return 0;
}
```

4. 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

4. Ejercicio 3. Entrada/Salida

- Realizad un programa que calcule el incremento de un salario de un empleado por un porcentaje especificado. Tanto salario como porcentaje se piden al usuario.

4. Ejercicio 3. Entrada/Salida

- Realizad un programa que calcule el incremento de un salario de un empleado por un porcentaje especificado. Tanto salario como porcentaje se piden al usuario.

```
#include <stdio.h>

int main() {
    float salario, incremento;

    printf("Ingrese salario: ");
    scanf("%f", & salario);
    printf("Ingrese incremento (número entre 0 y 1: ");
    scanf("%f", & incremento );

    printf("Salario inicial: %.2f\n", salario);

    salario *= (1 + incremento);

    printf("Salario después del incremento: %.2f\n", salario);

    return 0;
}
```

5. 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
$++n, n++$	$--n, n--$
$n += 1$	$n -= 1$
$n = n + 1$	$n = n - 1$

6. 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 expresion`

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

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

6. Ejercicio 4. Operadores relacionales

- 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.

6. Ejercicio 4. Operadores relacionales

- 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)
```

7. 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.

7. 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

7. Ejercicio 5. Operadores lógicos

- 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

7. Ejercicio 5. Operadores lógicos

```
#include <stdio.h>

void main() {
    float op1=0,op2=0,op3=0,op4=0;
    int opLogica1=0,opLogica2=0,opLogica3=0,opLogica4=0;

    printf("Introduzca cuatro valores en coma flotante \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. Ejercicio 6. Operadores relacionales

- Generar un programa que pida un carácter por pantalla, y que compruebe si es una letra tanto mayúscula como minúscula. 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.

7. Ejercicio 6. Operadores relacionales

- Generar un programa que pida un carácter por pantalla, y que compruebe si es una letra tanto mayúscula como minúscula. 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.

```
#include <stdio.h>

void main (int argc, char **argv){

    char Caracter='\0';
    int EsCaracter=0, int EsMayuscula=0, int EsMinuscula=0;

    printf("Introduzca una caracter\n");
    scanf("%c", &Caracter);

    EsMayuscula = (('A' <= Caracter) && (Caracter <= 'Z'));
    printf("¿Es la letra mayuscula? (SI=1 / NO = 0)  %d\n",EsMayuscula);

    EsMinuscula = (('a' <= Caracter) && (Caracter <= 'z'));
    printf("¿Es la letra minuscula? (SI=1 / NO = 0)  %d\n",EsMinuscula);

    EsCaracter = (('A' <= Caracter) && (Caracter <= 'Z') || ('a' <= Caracter) && (Caracter <= 'z'));
    printf("¿Es una letra? (SI=1 / NO = 0) %d\n",EsCaracter);

}
```


8. 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 2^n .

8. Ejercicio 7: operadores manipulación de bits

- 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.

8. Ejercicio 7: operadores manipulación de bits

- 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.

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

Usando operadores de desplazamiento

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

tercerBit= 1 << 2; /* desplazar 2 posiciones el 00000001 */
               /* en tercerBit quedaría el 00000100 */

esUno = tercerBit & dato; // hacemos un "and" bit a bit
esUno = esUno >> 2;
printf("El resultado del and bit a bit es:%d\n", esUno);
```

8. Ejercicio 7: operadores manipulación de bits

- 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.

```
#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;
esUno = esUno >> 2;
printf("El resultado es:%d\n", esUno);
```

Usando máscaras
de bit

8. Ejercicio 8: operadores manipulación de bits

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

8. Ejercicio 8: operadores manipulación de bits

- Crear un programa 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. El exponente debe ser mayor que cero. No se puede usar el operador "*" (multiplicar).

```
#include <stdio.h>

void main(){
    int num,res,exponente;

    printf("Ingrese un número entero: ");
    scanf("%d", &num);
    printf("Ingrese el exponente: ");
    scanf("%d", &exponente);

    res=num<<exponente;
    printf("El resultado es %d: ",res);
}
```

9. 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. Ejercicio 9: operadores manipulación de bits

- Define una variable tipo carácter, entera y float. Asígnales un valor y escribe su valor y la dirección de memoria que ocupan.

9. Ejercicio 9: operadores manipulación de bits

- Define una variable tipo carácter, entera y float. Asígnales un valor y escribe su valor y la dirección de memoria que ocupan.

```
#include <stdio.h>

void main(){
    char Character = 'a';
    int Entero = 5;
    float Racional = 7.7;

    printf ("Caracter vale %c y ocupa la dirección %x \n",Character,&Character);
    printf ("Entero vale %d y ocupa la dirección %x \n",Entero,&Entero);
    printf ("Racional vale %f y ocupa la dirección %x \n",Racional,&Racional);
}
```

10. 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 luego 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.

11. 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.

11. Conversiones de tipos

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

11. Ejercicio 10: Conversiones de tipos

- Convertir un numero entero introducido por teclado a carácter e imprimirlo por pantalla.

11. Ejercicio 10: Conversiones de tipos

- Convertir un numero entero introducido por teclado a carácter e imprimirlo por pantalla

```
#include <stdio.h>

void main (int argc, char **atgv[]){

    int ent = 0;
    char car = '\0';

    printf ("Introduce un entero: \n");
    scanf ("%d",&ent);
    printf ("El hex de ent es: %x\n",ent);

    car = (char)ent;
    printf ("%c\n", car);
    printf ("El hex de car es: %x\n",car);

}
```

A nivel de bits ¿Qué ocurre?

Al realizar el casting, `car = (char)ent`, en `car` se queda el byte menos significativo del entero.

11. Ejercicio 11: Conversiones de tipos

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

11. Ejercicio 11: Conversiones de tipos

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

```
#include <stdio.h>

void main (int argc, char **argv){

    int Ent = 0;
    float Real =0.0;

    printf ("Introduce un real: \n");
    scanf ("%f",&Real);
    printf ("El numero introducido es el real es: %f\n",Real);
    printf ("El hex de Real es: %x\n",Real);

    Ent = (int)Real;
    printf("%d\n", Ent);
    printf ("El hex de Ent es: %x\n",Ent);
}
```

A nivel de bits ¿Qué ocurre?

Al realizar el casting, `Ent = (int)Real`, no trunca a nivel de bytes sino trunca a nivel de cifras y se queda con la parte entera

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

11. Ejercicio 12: Conversiones de tipos

- Declara dos enteros (Entero1 y Entero2) y dos reales (Real1 y Real2) y realiza las siguientes operaciones, explicando el resultado.
 - $\text{Entero1} = \text{Real1};$
 - $\text{Real2} = \text{Entero } 2;$
 - $\text{Real1} = \text{Entero1} + \text{Real1};$
 - $\text{Real2} = \text{Entero2} / \text{Entero1};$
 - $\text{Real2} = \text{Entero2} / \text{Real1};$

11. Ejercicio 12: Conversiones de tipos

- Declara dos enteros (Entero1 y Entero2) y dos reales (Real1 y Real2) y realiza las siguientes operaciones, explicando el resultado.
 - $\text{Entero1} = \text{Real1};$
 - $\text{Real2} = \text{Entero2};$
 - $\text{Real1} = \text{Entero1} + \text{Real1};$
 - $\text{Real2} = \text{Entero2} / \text{Entero1};$
 - $\text{Real2} = \text{Entero2} / \text{Real1};$

```
#include <stdio.h>

void main (int argc, char **atgv[]){

    int Entero1 =1,Entero2 =2;
    float Real1 = 3.3, Real2 = 5.5;

    Entero1 = Real1;
    printf ("\n Despues de la asignacion de un Real a un entero El hex de Entero1 es: %d\n",Entero1);
    printf ("\n Despues de la asignacion de un Real a un entero El hex de Entero1 es: %x\n",Entero1);
    //La asignacion de un Real a un entero no trunca a nivel de bytes sino que transforma
    //trunca a nivel de cifras y se queda con la parte entera

    Real2 = Entero2;
    printf ("\n Despues de la asignacion de un entero a un Real El float de Real es: %f\n",Real2);
    printf ("\n Despues de la asignacion de un entero a un Real El hex de Real es: %x\n",Real2);
    // La asignacion de un entero a un Real asigna el valor del entero al Real sin conversion

    Real1 = Entero1 + Real1;
    printf ("\n Despues de la asignacion de un entero a un Real El float de Real es: %f\n",Real1);
    printf ("\n Despues de la asignacion de un entero a un Real El hex de Real es: %x\n",Real1);
    // Realiza la suma de igual modo que si fuera coma flotante

    Real2 = Entero2 / Entero1 ;
    printf ("\n Despues de la asignacion de un entero a un Real El float de Real es: %f\n",Real2);
    printf ("\n Despues de la asignacion de un entero a un Real El hex de Real es: %x\n",Real2);
    // Entero2 / Entero1 es una división de enteros y luego se realiza la asignación a un Real

    Real2 = Entero2 / Real1 ;
    printf ("\n Despues de la asignacion de un entero a un Real El float de Real es: %f\n",Real2);
    printf ("\n Despues de la asignacion de un entero a un Real El hex de Real es: %x\n",Real2);
    // Es una division real ya que Real1 lo es que se asigna a un real

}
```

11. Ejercicio 13: Conversiones de tipos

- 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

11. Ejercicio 13: Conversiones de tipos

- 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;
```

- $b * c \rightarrow$ 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
- $(b * c) / d \rightarrow$ 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**.

11. Ejercicio 13: Conversiones de tipos

- 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;
```

- $b * c \rightarrow$ b se convierte en int. Las variables b y c son de tipo int, se ejecuta el producto y el resultado es un int.
- $(b * c) / d \rightarrow$ 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**.

11. Ejercicio 13: Conversiones de tipos

- 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



CENTRO UNIVERSITARIO
DE TECNOLOGÍA Y ARTE DIGITAL