Operadores, expresiones

Operadores

- Nos permiten modificar y manipular información
- Son símbolos que representan una operación de uso frecuente, como por ejemplo la suma:
  - 3 + 4

Operador de asignación

- Utilizamos el operador de asignación "=" para asignar un determinado valor a una variable (o constante)

```
const float RATE = 0.1;
int income = 5;
float tax = income * RATE;

RATE = 0.05;          //NO! RATE es const
4 = income + 64;      //NO! 4 no es una variable
```

Operadores aritméticos

- “+”, “-”, “*”, “/”, “%” (aritmética modular)

```
someValue = num1 + num2;   // suma

someValue = num1 - num2;   // resta

someValue = num1 * num2;   // producto

someValue = num1 / num2;   // división

someValue = num1 % num2;   // modulo (resto)
```

Promoción de tipos

```
int integer1;
float float1;

float1 + integer1 // da un float
float1 - integer1 // da un float
float1 * integer1 // da un float
float1 / integer1 // da un float
integer1 / float1 // da un float
float1 % integer1 // no permitido
integer1 % float1 // no permitido
```

Ejemplo 1

```
float cels;
int fahr;


cels = (5/9)*(fahr-32);
cels = (5.0/9)*(fahr-32);
```

Ejemplo 2

```
float average_age;
int total_of_ages = 100;
int num_people = 6;


average_age = total_of_ages /
  num_people;
average_age =
  static_cast<float>(total_of_ages) /
  num_people;
```

Aritmética modular

- a % b nos da el resto de dividir a entre b
  - 4%7 es 4 (ya que 4/7 es 0 con resto 4)
  - 7%3 es 1 (ya que 7/3 es 2 con resto 1)
  - 27%3 es 0 (ya que 27/3 es 9 con resto 0)
- Por ejemplo, para calcular el dígito de las decenas de un número:

```
int tens_digit;
tens_digit = (numero%100)/10;
```

Operadores incremento y decremento

- val = val + 1;
  - val++; o
  - ++val;
- val = val – 1;
  - val--; o
  - --val;

Pre- vs. Post-

```
int val = 6;
int num;
num = ++val;

int val = 6;
int num;
num = val++;
```

Otros operadores abreviados

```
x += y;      // equivalente a  x = x + y;
x -= y;      // equivalente a  x = x - y;
x /= y;      // equivalente a  x = x / y;
x *= y;      // equivalente a  x = x * y;
x %= y;      // equivalente a  x = x % y;
```

Expresiones lógicas

- Son expresiones (normalmente booleanas) que serán evaluadas a true o false
- false se interpreta como cero y cero se interpreta como false
- true se interpreta como uno y cualquier número distinto de cero se interpreta como true

Operadores relacionales

- “<” menor que
- “<=” menor o igual que
- “>” mayor que
- “>=” mayor o igual que
- “==” igual que
- “!=” distinto de

Ejemplo

```
short val = 5;
short num = 8;
short bob = 0;

(val <= num);              // true
(num % val > bob);         // true (3 > 0)
(val == num);              // false
(num != (num / val));      // true
(num - val)                // se evalúa a true
(bob)                      // se evalúa a false
```

Operadores lógicos

- And lógico: “&&”
  - (a && b) es true si a y b son true
- Or lógico: “||”
  - (a || b) es true si a o b son true
- Not lógico: “!”
  - !a es true si a es false
- Nota: “&&” tiene precedencia sobre “||”

Ejemplo

```
short val = 5;
short num = 8;
short bob = 0;

((val == num) || (!bob)) && ((num - val) != (bob + 3));
```

Operadores de bits

- Operan con los bits de los operandos, que serán de tipo int, short o char (sean unsigned o no)
- And de bits: “&”
- Or de bits: “|”
- Or exclusivo de bits (XOR): “^”
- Not de bits: “~”
- Nota: “&” tiene precedencia sobre “|” y “^”
- Desplazamiento de bits a la izquierda: “<<”
- Desplazamiento de bits a la derecha: “>>”

Ejemplo

```
unsigned short a, b, c, d;

a = 0x5555;
b = 0xAAAA;
c = 0x00FF;

d = (a << 8) | (((~b) ^ c) << 4);
// cual es el valor de d en este punto?

d &= c;
// cual es el valor de d en este punto?
```