Operators and Type Casting

Expressions

- Combination of operators and operands
- Appear on the right side of an assignment statement

Operators

- Depending on the number of operand, operators can be-
  - Unary (-a)
  - Binary (a-b)
  - Ternary (later)
- Depending on the functionality, operators can be-
  - Arithmetic
  - Bitwise
  - Assignment
  - Relational
  - Logical
  - Others
- Operators containing two symbols can not be separated by space.

Arithmetic operators

Sign	Meaning	Type	Comments
+	Plus	Binary	
-	Minus	Binary	
*	Multiply	Binary	
/	Division	Binary	
%	Modulus	Binary	Operators can only be integer
++	Increment	Unary	
--	Decrement	Unary	
-	Unary negation	Unary	

Example

- `count=count*num+88/val-19%count;`
- `char x,y;`
- `x='a';`
- `y='b';`
- `int z=x+y;`

Increment and Decrement Operator

- Postfix operator
  - n++, n--
- Prefix operator
  - ++n, --n

Bitwise Operators

- These operators are used for bitwise logic operations.
- The operands must be integer values

'&'	bitwise AND	binary
'^'	bitwise XOR	binary
'|'	bitwise OR	binary
'~'	1's complement	unary
'!'	bitwise NOT	unary
'<<'	left shift	binary
'>>'	right shift	binary

Bitwise Operator

Operator	Description	Example
&	Bitwise AND	101 & 110 = 100
|	Bitwise OR	100 & 001 = 101
^	Bitwise XOR (exclusive OR)	110 & 101 = 011
~	1's complement	~100 = 011

Bitwise Operator (AND)

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise Operator (OR)

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise Operator (XOR)

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

Assignment Operators

- These operators assign the value of the expression on the right to the variable on the left

  '=' assign binary

- Shortcuts
  - a += b; means a = a + b;

  '+=', '- =', '*=', '/=', '%=',

  '&=', '|=', '^=', '<<=', '>>='

Relational Operators

- These operators are used for comparison. The result is boolean.

'<'	less than	binary
'>'	greater than	binary
'<='	less/equal	binary
'>='	greater/equal	binary
'=='	equal	binary
'!='	not equal	binary

Logical Operators

- These operators are evaluating logical expressions.
- The result is boolean

'&&'	logical AND	binary
'||'	logical OR	binary
'!'	logical NOT	unary

Operator Precedence

- If there are a chain of operations, then C defines which of them will be applied first.
- *, / and % are higher in precedence that + and -
- Precedence can be altered by using parentheses
  - Innermost parentheses evaluated first
- For example-

  6+4/2 is 8
  - because '/' has precedence over '+'
  - if we want the '+' to work first, we should write-

    (6+4)/2

Example

- Stepwise evaluation of the expression x=7/6*4+3/5+3
  - x=7/6*4+3/8+3
  - x=1*4+3/5+3 operation: /
  - x=4+3/5+3 operation: *
  - x=4+0+3 operation: /
  - x=4+3 operation: +
  - x=7 operation: +
- All the operators associate from left to right except for assignment operators

Type Conversion

- C allows mixing of types
- Integral promotion
  - During evaluation of an expression
  - `'A'+2`
- Type promotion
  - Converts all operands "up" to the type of the largest

Type conversion

- Operands that differ in type may undergo type conversion
- In general the result will be expressed in the highest precision possible
  - int i=7;
  - float f=5.5;
  - i+f : 12.5

Type Conversion

```
#include <stdio.h>
int main() {
    int i;
    float f;
    i=10;
    f=23.25;
    printf ("%f \n", i*f);
    return 0;
}
```

Type Conversion in assignment

- Type of right side is converted to the type if the left

```
#include <stdio.h>
int main() {
    int i;
    char c;
    i=1111;
    c=i;
    printf ("%d, %c \n", c, c);
    return 0;
}
```

Output: W, 87

Type Conversion

- Loss of precision

```
#include <stdio.h>
int main() {
    double f;
    f=7/2;
    printf ("%lf \n", f);
    return 0;
}
```

Type Cast

```
#include <stdio.h>

int main() {
    double f;
    f=7/2.0;
    printf ("%lf \n", f);
    return 0;
}
```

Type Cast

```
#include <stdio.h>

int main() {
    double f;
    f=7.0/2;
    printf ("%lf \n", f);
    return 0;
}
```

Type Cast

- Value of an expression can be converted to a different data type if desired.
- Temporary type change
- *(data type) expression*

  - int a=20, b = 8;
  - float f=a/b; // not correct result
  - float f=(float)a/b; // or the following
  - float f = a/ (float)b;

Type Cast

```
#include <stdio.h>

int main() {
    double f;
    f=(double)7/2;

    printf ("%lf \n", f);
    printf ("%d \n", (int)f);

    return 0;
}
```

Type Cast

```
#include <stdio.h>

int main() {
    double f;
    f=7/(double)2;

    printf ("%lf \n", f);
    printf ("%d \n", (int)f);

    return 0;
}
```

Type Cast

- Loss of precision

```
#include <stdio.h>

int main() {
    double f;
    f=(double)(7/2);

    printf ("%lf \n", f);
    printf ("%d \n", (int)f);

    return 0;
}
```

7/2=5
(double)(7/2)=(double)(5)=5.0