Ett uttryck har någon av formerna:

```
x op y
op x
x op
primärt uttryck
```

Exempel på uttryck är:

```
tal1    381    2.5    'A'    "Hej"    (i + j)
i + j      p = q          ++k      r -= s
n--        a + b * c      381      k == n
```

Aritmetiska uttryck:

```
t1 + t2          vardet + 5.2          1 - n          x - y
p * q            100 * antal           i / 10         x / y

+ 6          + n          - 3.7          + y

i / j            // heltalsdivision

7 / 2            // får värdet 3

12 % 5           // får värdet 2


- 2 + 4 / 2 * 3          // får värdet 4


(- 2 + 4) / 2 * 3        // får värdet 3
```

math.h

`sin`	`cos`	`tan`	argumentet anges i radianer
`asin`	`acos`	`atan`	ger arcsin etc.
`sinh`	`cosh`	`tanh`	hyperboliska funktioner
`exp`	ger e upphöjt till argumentet		
`log`	ger naturliga logaritmen ln		
`log10`	ger den vanliga logaritmen (med basen 10)		
`sqrt`	ger kvadratroten		
`ceil`	ger minsta hela tal som är större än eller lika med argumentet		
`floor`	ger största hela tal som är mindre än eller lika med argumentet		
`fabs`	ger absolutvärdet av argumentet		

++ --

```
antal = antal + 1;
rest  = rest - 1;

++i
--n
tal++
kvar--

++(i + j)     /* FEL! */
255--         /* FEL! */


++antal;
--rest;

antal++;
rest--;
```

```
i = 4; j = 7;
a = ++i * --j;      /* a får värdet 30 */
i = 4; j = 7;
b = i++ * j--;      /* b får värdet 28 */


i++ + i * j;        /* Olämpligt uttryck! */
```

Jämförelseoperatorer:

```
<       >       <=      >=

==      !=
```

Några exempel är:

```
x < y       a > 3.5       i + j <= 4       n >= k / i
x == y      i != 0        a * b + c == x - y
```

Ger värdet 1 för *sann* och värdet 0 för *falsk*. Typ: **`int`**.

```
if (i = 0)  { /* VANLIGT FEL! */
   printf("Noll");
}
else {
   printf("Inte noll");
}
```

Texten `Inte noll` kommer här att skrivas ut, oberoende av vilket värde i har.

Logiska operatorer:

```
aktiv && varm                    aktiv || klar
!aktiv                           i < 0 && j == 4
n == 100 && klar                  temp < 100 || !varm
!(n > 7)                         !(i == 0 && varm)
x > 0 || !(i == 2 && j < 4)
```

`a + b > c * d && e == f` tolkas som `((a + b) > (c * d)) && (e == f)`

`i || j && k` tolkas som `i || (j && k)`

`! aktiv && klar` tolkas som `(! aktiv) && klar`

För operatorerna `&&` och `||` gäller:

Den vänstra operanden räknas alltid ut först och högeroperanden beräknas bara om det är nödvändigt.

Villkorsoperatorn

```
if (x > y) {
   z = x;
}
else {
   z = y;
}
```

kan skrivas:

```
   z = (x > y) ? x : y;
```

```
/* översättning från stora till små bokstäver */
i = 0;
while ((c = s[i++]) != '\0') {
   putchar((c>='A' && c<='Z') ? c + 'a' - 'A' : c);
}
```

Bit-operatorer

~	negation, bit för bit
<<	vänsterskift
>>	högerskift
&	OCH, bit för bit
^	exklusiv ELLER, bit för bit
|	ELLER, bit för bit

```
c1 = 5;           /* c1 har bitmönstret 00000101 */
c2 = ~c1;         /* c2 får bitmönstret 11111010 */

c1 = 5;           /* c1 har bitmönstret 00000101 */
c2 = 6;           /* c2 har bitmönstret 00000110 */
c3 = c1 & c2;     /* c3 får bitmönstret 00000100 */
c4 = c1 | c2;     /* c4 får bitmönstret 00000111 */
c5 = c1 ^ c2;          /* c5 får bitmönstret 00000011 */

c6 = c1 && c2;    /* c6 får bitmönstret 00000001 */
c7 = c1 || c2;    /* c7 får bitmönstret 00000001 */
c8 = ! c1;        /* c8 får bitmönstret 00000000 */
c9 = ~ c1;        /* c9 får bitmönstret 11111010 */


c1 = 5;            /* c1 har bitmönstret 00000101 */
c2 = c1 << 3;      /* c2 får bitmönstret 00101000 */
c3 = c1 << 6;      /* c3 får bitmönstret 01000000 */
c3 = c1 >> 2;      /* c3 får bitmönstret 00000001 */

i = -3;          /* i har bitmönstret 1111111111111101 */
j = i >> 1;      /* j får bitmönstret ?111111111111110 */
```

Tilldelningsoperatorer:

```
=       *=      /=      %=      +=    -=
<<=     >>=     &=      ^=      |=
```

Exempel:

```
i = 0        x = y * z        rad[i] = 'A'

i = j = k = l = m = 0;     // multipel tilldelning
```

Sammansatt tilldelning:

```
i = i + k;
y = y - x;
```

kan skrivas:

```
i += k;
y -= x;
```

Uttrycket

op1 **op**= *op2*

är ekvivalent med

op1 = *op1* **op** *op2*

sizeof-operatorn

```
sizeof uttryck
sizeof (typnamn)
```

Ger operandens längd, uttryckt i antal bytes.

```
sizeof(unsigned char)
sizeof(signed char)
sizeof c
sizeof(i + j)
```

Antalet element i ett fält :

```
sizeof f / sizeof f[0]
```

sizeof används ofta vid anrop av standardfunktionen `malloc` :

```
malloc(100 * sizeof (int))
```