

C: an introduction

Expressions

1

Program: building blocks

- Variables
 - Store data (input, intermediate values, results)
- Expressions
 - Manipulate variables
- Control structures
 - Make decisions (if) or repeat (for, while) statements
- Functions
 - Combine expressions and structures for parameterization and re-use

2

Operators: overview

- arithmetic operators
- relational operators
- logical operators
- bitwise operators
- assignment operators
- incremental operators
- conditional operator

<https://github.com/gjbex/training-material>

Arithmetic Operators

- Arithmetic operators are
 - + plus
 - minus
 - * multiply
 - / divide
 - = assignment
 - % modulus (remainder after division)
- The first 5 are valid for integer and floating-point types.
- The % is valid only for integer types (including `char`).

Arithmetic Expressions

`3.0 / 5.0` - equals 0.6
`3 / 5` - integer division truncates, equals 0
`17 / 6` - equals 2
`18 % 7` - equals 4

`2*7 + 5*9` - equals 14 + 45: 59

File: arithmetic_1.c

Hands-on: change type `int` into `double`

```
1 #include <stdio.h>
2
3 /*
4  * arithmetic_1.c
5  */
6
7 int main ()
8 {
9     int var1 = 10;
10    int var2 = 2;
11    int var3 = 35;
12    int var4 = 8;
13    int result;
14
15    result = var1 + var2;
16
17    printf ("Sum of var1 and var2 is %d\n", result);
18    result = var3 * var3;
19    printf ("Square of var3 is %d\n", result);
20    result = var2 + var3 * var4; /* precedence */
21    printf ("var2 + var3 * var4 = %d\n", result);
22    result = var3 % var1;
23    printf ("var3 %% var1 is %d\n", result);
24
25    return 0;
26 }
27
28
```

```
frankvp@CRD-L-08004:~/Expressions$ gcc arithmetic_1.c -o arithmetic_1
frankvp@CRD-L-08004:~/Expressions$ ./arithmetic_1
Sum of var1 and var2 is 12
Square of var3 is 1225
var2 + var3 * var4 =282
var3 % var1 is 5
frankvp@CRD-L-08004:~/Expressions$
```

```

1 #include <stdio.h>
2
3 /*
4  arithmetic_1_double.c
5  */
6
7 int main ()
8 {
9     double var1 = 10;
10    double var2 = 2;
11    double var3 = 35;
12    double var4 = 8;
13    double result;
14
15    result = var1 + var2;
16
17    printf ("Sum of var1 and var2 is %f\n", result);
18    result = var3 * var3;
19    printf ("Square of var3 is %f\n", result);
20    result = var2 + var3 * var4; /* precedence */
21    printf ("var2 + var3 * var4 =%f\n", result);
22    /* result = var3 % var1;
23    printf ("var3 %% var1 is %d\n", result);
24    */
25    return 0;
26 }
27
28

```

```

frankvp@CRD-L-08004:~/Expressions$ gcc arithmetic_1_double.c -o arithmetic_1_double
frankvp@CRD-L-08004:~/Expressions$ ./arithmetic_1_double
Sum of var1 and var2 is 12.000000
Square of var3 is 1225.000000
var2 + var3 * var4 =282.000000
frankvp@CRD-L-08004:~/Expressions$

```

KU LEUVEN

7

Arithmetic Evaluation

- Precedence and order of evaluation.
- eg, $a + b * c$
- Order of evaluation from left to right.
- $*$, $/$ and $\%$ take precedence over $+$ and $-$, so that

$a + b * c$ is the same as

$a + (b * c)$

- Precedence table exists, but use brackets $()$ instead for clarity

KU LEUVEN

8

Incremental Operators

- Valid operators on integer or floating-point numbers.

- **Prefix**

`++x` is a shortcut for `x=x+1`

`--x` is a shortcut for `x=x-1`

`y=++x` is a short cut for `x=x+1; y=x;`
`x` is evaluated **after** it is incremented.

`y=--x` is a short cut for `x=x-1; y=x;`
`x` is evaluated **after** it is decremented.

Incremental Operators

- **Postfix**

`x++` is a short cut for `x=x+1`

`x--` is a short cut for `x=x-1`

`y=x++` is a short cut for `y=x; x=x+1`
`x` is evaluated **before** it is incremented.

`y=x--` is a short cut for `y=x; x=x-1`
`x` is evaluated **before** it is decremented.

Incremental Operators

- ++ as postfix-operator `n++`
 - `x` is incremented by 1, after using the (old) value in the expression

```
x = 5;
y = ++x;    /* y is 6, x is 6 */

x = 5;
y = x++;    /* y is 5, x is 6 */
```
- `x++`; are identical `++x`; (as single statement)
- only applicable to variables
 - `(i+j)++` is not allowed

- *increment.c*

```
1 /*
2 increment.c
3
4 Usage of pre-fixing and post-fixing the increment operator
5 */
6
7 #include <stdio.h>
8
9 void main()
10 {
11
12
13     int x = 10;
14
15     printf("Value of x after pre-fixing ++ is %d\n", ++x);
16
17     printf("Value of x after post-fixing ++ is %d\n", x++);
18
19     printf("be careful \n");
20     printf("What is this - x: %d - x++ : %d - ++x: %d \n", x, x++, ++x);
21
22
23 }
24
```

```
frankvp@CRD-L-08004:~/Expressions$ gcc increment.c -o increment
frankvp@CRD-L-08004:~/Expressions$ ./increment
Value of x after pre-fixing ++ is 11
Value of x after post-fixing ++ is 11
be careful
What is this - x: 14 - x++ : 13 - ++x: 14
frankvp@CRD-L-08004:~/Expressions$
```

Relational Operators

- Relational operators are
 - > greater-than
 - < less-than
 - >= greater-than-or-equal-to
 - <= less-than-or-equal-to
 - == equal-to
 - != not-equal-to
- These operators are valid for integer and floating-point types.
- Evaluate to 1 if TRUE, and 0 if FALSE

`3.2 < 7` equals **1**, and `x != x` equals **0**

Logical Operators

- Logical operators are
 - && AND
 - || OR
 - ! NOT
- && and || connect multiple conditional expressions.
- ! negates a conditional expression (non-zero becomes 0, zero becomes 1).

Relational and Logical Expressions

```
int a=1, b=2, c=3, d=3;
a < b && b < c && c < d    /* FALSE */
a < b && b < c && c <= d   /* TRUE */
(a < b && b < c) || c < d /* TRUE */
a && !b /* FALSE */
```

- `&&` and `||` are evaluated left-to-right and, once the result of TRUE or FALSE is known, evaluation stops – leaving the remaining expressions unevaluated.

boolean

- ANSI C has no Boolean type
- C99 standard supports boolean data type
- To use `bool`, `true`, and `false` include `stdbool.h`
- Integral types may be treated as boolean expressions
 - 0 considered false
 - Any non-zero is considered true

logical.c

```
1#include <stdio.h>
2
3/*
4logical.c
5Demonstrate logical and relational operators
6*/
7
8int main(void)
9{
10    int a=1, b=2, c=3, d=3;
11    int e;
12
13    printf("a = %d, b = %d, c = %d, d = %d\n", a, b, c, d);
14    printf("(a < b && b < c && c < d) = %d\n", a < b && b < c && c < d);
15    printf("(a < b && b < c && c <= d) = %d\n", a < b && b < c && c <= d);
16    printf("((a < b && b < c) || c < d) = %d\n", (a < b && b < c) || c < d);
17
18
19    e = (a == b);
20    printf("e = (a == b): %d \n", e);
21    e = (c == d);
22    printf("e = (c == d): %d \n", e);
23
24    return 0;
25}
```

```
frankvp@CRD-L-08004:../Expressions$ gcc logical.c -o logical
frankvp@CRD-L-08004:../Expressions$ ./logical
a = 1, b = 2, c = 3, d = 3
(a < b && b < c && c < d) = 0
(a < b && b < c && c <= d) = 1
((a < b && b < c) || c < d) = 1
e = (a == b): 0
e = (c == d): 1
frankvp@CRD-L-08004:../Expressions$
```

KU LEUVEN

17

leapyear.c

```
1#include <stdio.h>
2
3/*
4leapyear.c
5Determine whether year is a leap-year
6*/
7
8int main(void)
9{
10    const int MIN_YEAR = 1828;
11    const int MAX_YEAR = 3003;
12    int year;
13
14    printf("Enter a year between %d and %d: ", MIN_YEAR, MAX_YEAR);
15    scanf("%d", &year);
16    if (year < MIN_YEAR || year > MAX_YEAR) {
17        printf("Error: invalid year!!\n");
18        return -1;
19    }
20
21    /* A leap year must be divisible by 4 but not by 100, except
22     * that years divisible by 400 are leap years */
23    if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)
24        printf("The year %d is a leapyear.\n", year);
25    else
26        printf("The year %d is not a leapyear.\n", year);
27}
```

```
frankvp@CRD-L-08004:../Expressions$ gcc leapyear.c -o leapyear
frankvp@CRD-L-08004:../Expressions$ ./leapyear
Enter a year between 1828 and 3003: 2000
The year 2000 is a leapyear.
frankvp@CRD-L-08004:../Expressions$ ./leapyear
Enter a year between 1828 and 3003: 2001
The year 2001 is not a leapyear.
frankvp@CRD-L-08004:../Expressions$ ./leapyear
Enter a year between 1828 and 3003: 2100
The year 2100 is not a leapyear.
frankvp@CRD-L-08004:../Expressions$
```

KU LEUVEN

18

Bitwise Operators

- Used to manipulate individual bits inside an integer.
- Bitwise operators are
 - & bitwise AND
 - | bitwise OR
 - ^ bitwise XOR
 - << left shift
 - >> right shift
 - ~ one's complement (bitwise NOT)
- Beware:
 - & is not &&
 - | is not ||

```
1 /*
2 bitoper_2.c
3 taken from https://www.programiz.com/c-programming/bitwise-operators
4
5 12 = 00001100 (In Binary)
6 25 = 00011001 (In Binary)
7
8 */
9
10 #include <stdio.h>
11 int main(void)
12 {
13     int a = 12, b = 25;
14
15     printf("Output = %d \n", a&b);
16     printf("Output = %d \n", a|b);
17     printf("Output = %d \n", a^b);
18     printf("\n");
19
20     int num=210, i;
21     for (i=0; i<=2; ++i)
22         printf("Right shift by %d: %d\n", i, num>>i);
23
24     printf("\n");
25
26     for (i=0; i<=2; ++i)
27         printf("Left shift by %d: %d\n", i, num<<i);
28
29     return 0;
30 }
```

bitoper_2.c

```
frankvp@CRD-L-08004:~/Expressions$ gcc bitoper_2.c -o bitoper_2
frankvp@CRD-L-08004:~/Expressions$ ./bitoper_2
Output = 8
Output = 29
Output = 21

Right shift by 0: 210
Right shift by 1: 105
Right shift by 2: 52

Left shift by 0: 210
Left shift by 1: 420
Left shift by 2: 840
frankvp@CRD-L-08004:~/Expressions$
```

Assignment Operators

- Assignment operators - for example,

`a += b;` is equivalent to

`a = a + b;`

`x *= y+1;` is equivalent to

`x = x * (y+1);`

- Assignment also with other arithmetic operators: `+`, `-`, `*`, `/`, `%`

factorial.c

```
1#include <stdio.h>
2/*
3factorial.c
4Calculate the factorial of a non-negative integer.
5Result = n*(n-1)*(n-2)* ... *2*1
6Note, the factorial of 0 is 1.
7*/
8
9int main(void)
10{
11    int result=1, n;
12
13    /* Get user input */
14    printf("Enter a non-negative integer value: ");
15    scanf("%d", &n);
16    if (n < 0) {
17        printf("Error: number must be 0 or greater!\n");
18        return -1;
19    }
20
21    /* Calculate factorial */
22    while (n)
23        result *= n--;
24
25    printf("The factorial is %d\n", result);
26}
```

```
frankvp@CRD-L-08004:../Expressions$ gcc factorial.c -o factorial
frankvp@CRD-L-08004:../Expressions$ ./factorial
Enter a non-negative integer value: 6
The factorial is 720
frankvp@CRD-L-08004:../Expressions$
```