Problem Solving Through programming in C

Course Code:ONL1001

Expression and Operators

Ms. Shubhra dwivedi
Assistant Professor
School - SCOPE
VIT-AP Amaravati
Shubhra.d@vitap.ac.in

Elements of a program

- Literals → fixed data written into a program
- Variables & constants → placeholders (in memory) for pieces of data
- Types → sets of possible values for data
- Expressions → combinations of operands (such as variables or even "smaller" expressions) and operators. They compute new values from old ones.
- Assignments → used to store values into variables
- Statements → "instructions". In C, any expression followed by a semicolon is a statement

Elements of a program

- Control-flow constructs → constructs that allow statements or groups of statements to be executed only when certain conditions hold or to be executed more than once.
- Functions → named blocks of statements that perform a well-defined operation.
- Libraries → collections of functions.

Statement

- Statements are elements in a program which (usually) ended up with semi-colon (;)
  - e.g. below is a variables declaration statement

```
int a, b, c;
```

- Preprocessor directives (i.e. `#include` and `define`) are not statements. They don't use semi-colon

An expression statement is a statement that results a value

Some examples of expression	Value
• **Literal expression** e.g. 2, “A+”, ‘B’	The literal itself
• **Variable expression** e.g. Variable1	The content of the variable
• **arithmetic expression** e.g. 2 + 3 -1	The result of the operation

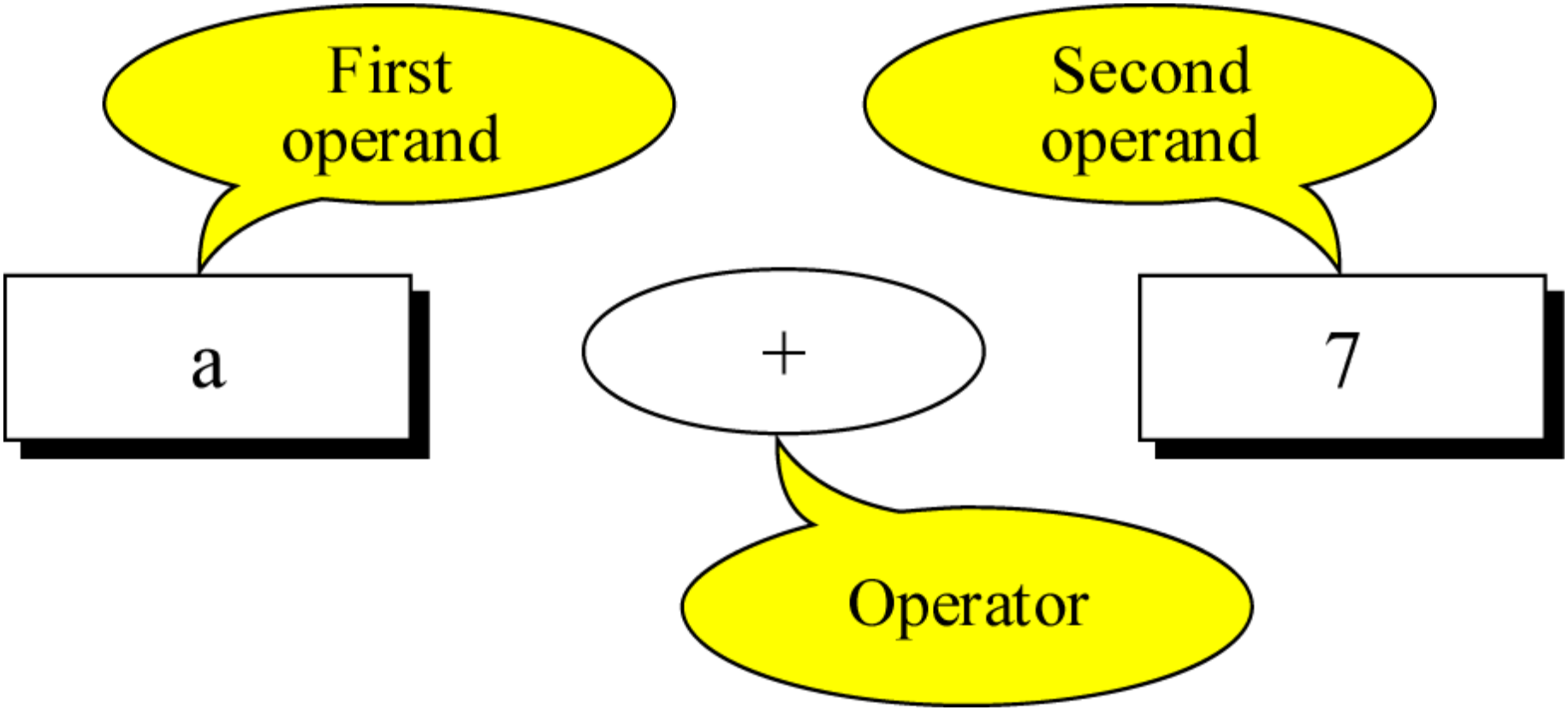
Operators

- Operators can be classified according to
  - the type of their operands and of their output
    - Arithmetic
    - Relational
    - Logical
    - Bitwise
  - the number of their operands
    - Unary (one operand)
    - Binary (two operands)
    - Ternary (three operands)

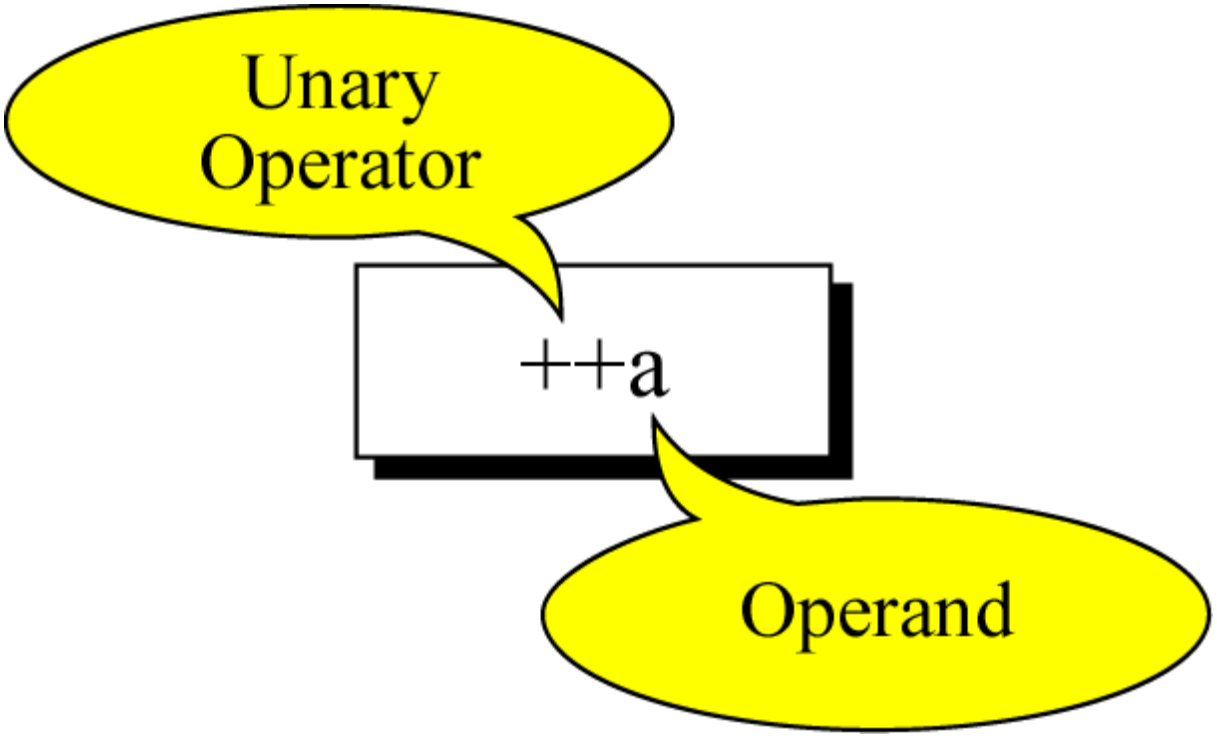
Operators in C

	Operator	Type
Unary operator	+ +, - -	Unary operator
Binary operator	+, -, *, /, %	Arithmetic operator
	<, <=, >, >=, ==, !=	Relational operator
	&&, ||, !	Logical operator
	&, |, <<, >>, ~, ^	Bitwise operator
	=, +=, -=, *=, /=, %=	Assignment operator
Ternary operator	?:	Ternary or conditional operator

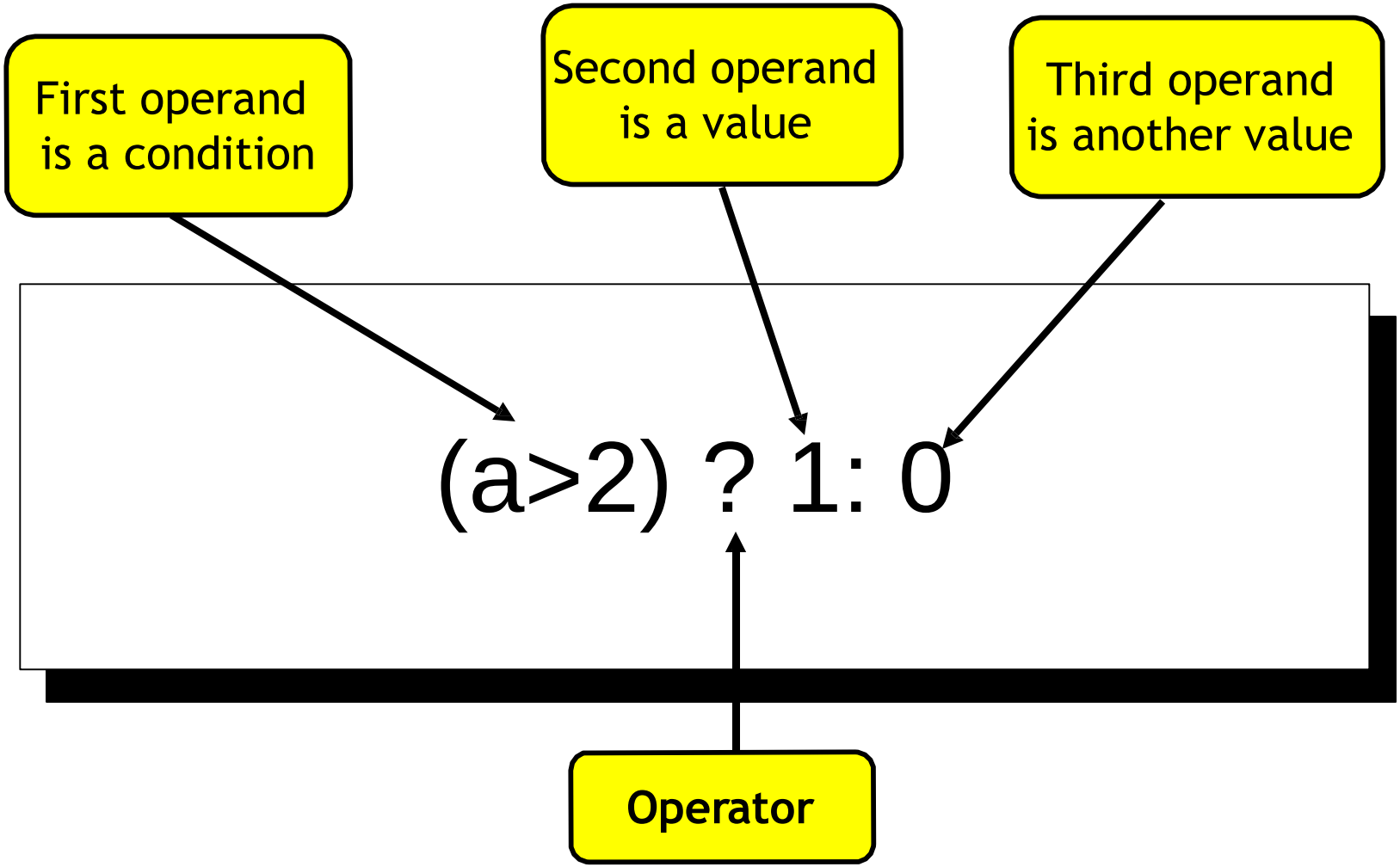
Binary expression



Unary Expression



Ternary Expression



Arithmetic operators

- They operate on numbers and the result is a number.
- The type of the result depends on the types of the operands.
- If the types of the operands differ (e.g. an integer added to a floating point number), one is "promoted" to other.
  - The "smaller" type is promoted to the "larger" one.

`char` → `int` → `float` → `double`

Example of promotion:

The result of the following “float division” is 2.5

5 / 2.0

Before the division process, 5 is promoted from integer 5 to float 5.0

The result of the following “integer division” is 2

5 / 2

There is no promotion occurred. Both operands are the same type.

```
#include<stdio.h>
 int main()
 {
   int a,c;
   char b;
   a = 1;
   b = 'a';
   c = a+b;
   printf("%d",c);
 }
```

Output: 98

In the above given example, variable a is of type int and variable b is of type char. When addition is done in line 8 and stored in variable c and when we print it (line 9), we get the output as 98. ASCII value of variable ‘a’ is 97 added with 1. Thus char variable b is implicitly type converted to integer.

```
#include<stdio.h>
 int main()
 {
   int a = 11;    // integer a
   char b = 'g';  // character b
   // b implicitly converted to int.
   // ASCII  value of ‘g' is 103
   a = a + b;
   // a is implicitly converted to double
   double c = a + 1.0;
   printf("a = %d\nc = %e\n", a,c);
   return 0;
 }
```

Output: a=114
C=1.150000e+02

In the above given example, variable c is of type double and variable a is of type int. In line 11, variable a is implicitly converted into double and added with value 1.0. The final output is stored in variable c.

```
#include<stdio.h>
 int main()
 {
   int a = 11;    // integer a
   char b = 'g';  // character b

   unsigned long int d = 1;
   float c = 3.0;
   double e = 1.0;
   long double f = 2.0;

   d = a + d;   // a is implicitly converted into
unsigned long int
   c = a + c;   // a is implicitly converted into
float
   e = a + e;   // a is implicitly converted into
double
   f = a + f;    // a is implicitly converted into
long double

   printf("unsigned long int d = %ld\nfloat c =
%e\ndouble e = %e\nlong double f =
%Lf\n",d,c,e,f);
   return 0;
 }
```

Output:

```
unsigned long int d = 12
float c = 1.400000e+01
double e = 1.200000e+01
long double f = 13.000000
```

In the given example, variable a in line 12 is implicitly converted into unsigned int since variable ‘d’ is of type unsigned long int, in line 13 variable ‘a’ is implicitly converted into type float since variable ‘c’ is of type float, in line 14 variable ‘a’ is implicitly converted into type double since variable ‘e’ is of type double, in line 15 variable ‘a’ is implicitly converted into type long double since variable ‘f’ is of type long double.

Integer Arithmetic

- When both the operands in a single arithmetic expression are integers, the expression is called an integer expression , and the operation is called integer arithmetic.
- During modulo division the sign of the result is always the sign of the first operand. That is
- -14 % 3 = -2
- -14 % -3 = -2
- 14 % -3 = 2

For example:

```
#include <stdio.h>
int main()
{
   int a = 3, b = -8, c = 2;
   printf("%d", a % b / c);
   return 0;
}
Output:1
```

```
#include <stdio.h>
int main()
{
  // a positive and b negative.
  int a = 3, b = -8;
  printf("%d", a % b);
  return 0;
}
Output: 3
```

```
#include <stdio.h>
int main()
{
  // a negative and b positive
  int a = -3, b = 8;
  printf("%d", a % b);
  return 0;
}
Output: -3
```

```
#include <stdio.h>
int main()
{
  // a and b both negative
  int a = -3, b = -8;
  printf("%d", a % b);
  return 0;
}
Output: -3
```

Real Arithmetic

- An arithmetic operation involving only real operands is called real arithmetic. If x and y are floats then we will have:
- 1) x = 6.0 / 7.0 = 0.857143
- 2) y = 1.0 / 3.0 = 0.333333
- The operator % cannot be used with real operands.

Mixed-mode Arithmetic

- When one of the operands is real and the other is integer, the expression is called a mixed-mode arithmetic expression and its result is always a real number.

Eg:
1) 15 / 10.0 = 1.5

Arithmetic operators: +, *

- **+** is the addition operator
- ***** is the multiplication operator
- They are both binary

Arithmetic operator: –

- This operator has two meanings:
  - subtraction operator (binary)

    e.g. `31 - 2`

  - negation operator (unary)

    e.g. `-10`

Arithmetic operator: /

- The result of integer division is an integer:

e.g. **`5 / 2 is 2, not 2.5`**

Arithmetic operator: %

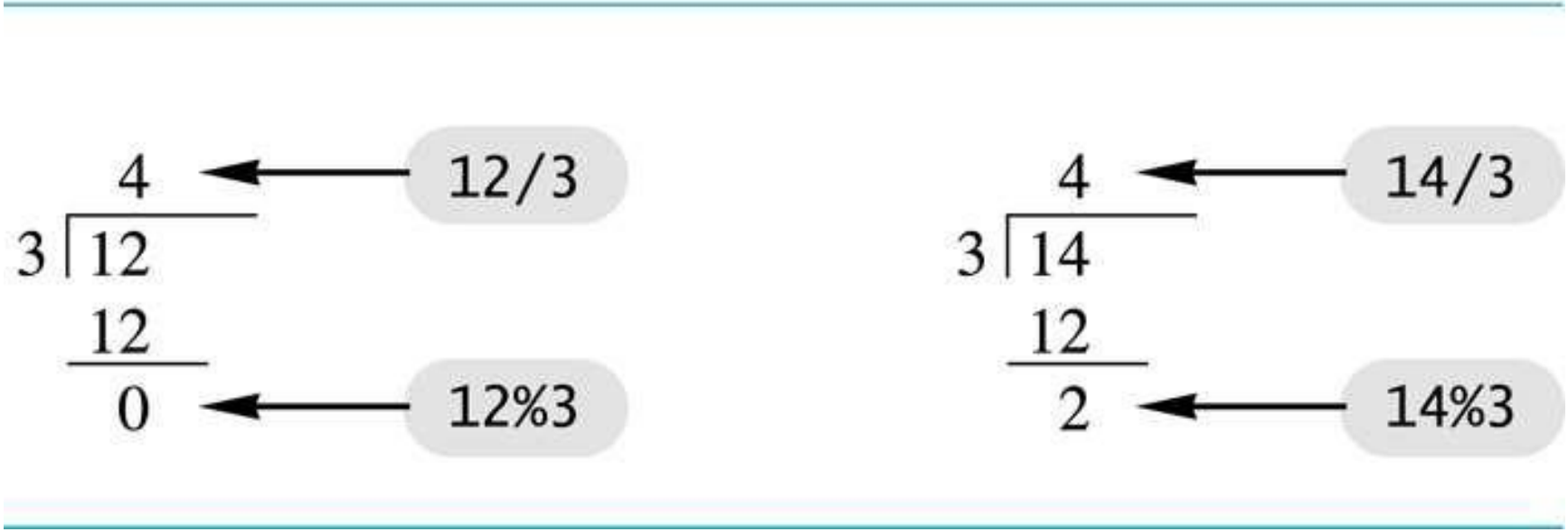
- The modulus (remainder) operator.
- It computes the remainder after the first operand is divided by the second

```
e.g. 5 % 2 is 1, 6 % 2 is 0
```

- It is useful for making cycles of numbers:
  - For an int variable x :

```
if   x is:  0  1  2  3  4  5  6  7  8  9 ...
(x%4) is:  0  1  2  3  0  1  2  3  0  1 ...
```

Integer Division



```c
#include <stdio.h>
int main (void)
{
    int a = 100;
    int b = 2;
    int c = 25;
    int d = 4;
    int result;

    result = a - b;      // subtraction
    printf ("a - b = %i\n", result);

    result = b * c;      // multiplication
    printf ("b * c = %i\n", result);

    result = a / c;      // division
    printf ("a / c = %i\n", result);

    result = a + b * c;   // precedence
    printf ("a + b * c = %i\n", result);

    printf ("a * b + c * d = %i\n", a * b + c * d);
    return 0;
}
```

Output

```
a - b = 98
b * c = 50
a / c = 4
a + b * c = 150
a * b + c * d = 300
```

```
#include <stdio.h>

int main (void)
{
    int   a = 25;
    int   b = 2;

    float c = 25.0;
    float d = 2.0;

    printf ("6 + a / 5 * b = %i\n", 6 + a / 5 *
b);
    printf ("a / b * b = %i\n", a / b * b);
    printf ("c / d * d = %f\n", c / d * d);
    printf ("-a = %i\n", -a);

    return 0;
}
```

Output:
6 + a / 5 * b = 16
a / b * b = 24
c / d * d = 25.000000
-a = -25