

CSE / ENGR 142 Programming I

Arithmetic Expressions

© 2000 UW CSE

1/7/00

C-1

Assignment Statement Review

```
double area, radius;
```

```
area = 3.14 * radius * radius;
```

assignment statement

expression

1/7/00

C-2

Why Study Expressions?

1. We need precise rules that define exactly what an expression means:

What is the value of $4 - 4 * 4 + 4$?

2. Arithmetic on a computer isn't always precise:

$(1.0 / 9.0) * 9.0$ could be 0.99999998213

3. Division of "int" type variables can give REALLY different results from what you probably expect:

$2 / 3$ is zero in C

1/7/00

C-3

Expressions

- Expressions are things that have **values**

- A **variable by itself** is an expression: *radius*

- A **constant by itself** is an expression: 3.14

- Often expressions are **combinations** of variables, constants, and operators.

```
area = 3.14 * radius * radius;
```

- The overall value of the expression is based on the data and operators specified.

- **Data** means the integer or floating-point constants and/or variables in the expression.

- **Operators** are things like addition, multiplication, etc.

1/7/00

C-4

The Big Picture

- In an assignment statement,
 - the expression (right hand side) is first **evaluated**,
 - then its value is **assigned to** (stored in) the **assignment variable** (left hand side).
- How this happens depends on the data **types** in the expression, the **operators**, and the **type** of the assignment variable.

```
my_int = int1 + int2;      int1  1  my_int    
                          int2  3  1/7/00  C-5
```

Unary and Binary

- **Binary**: operates on **two** things
 $3.0 * b$
zebra + giraffe
- **Unary**: operates on **one** thing
 -23.4
- C operators are unary or binary
- Then what about expressions like $a+b+c$?
 - Answer: this is two binary ops, in sequence

1/7/00

C-6

Expressions with *doubles*

REVIEW:

Doubles are floating-point values that represent real numbers within the computer.

Constants of type double:

0.0, 3.14, -2.1, 5.0, 6.02e23, 1.0e-3

not 0 or 17

Operators on doubles:

unary: -

binary: +, -, *, /

1/7/00

C-7

Expressions with *doubles*: Examples

double height, base, radius, x, c1, c2 ;

Sample expressions (not statements):

*0.5 * height * base*

*(4.0 / 3.0) * 3.14 * radius * radius * radius*

*- 3.0 + c1 * x - c2 * x * x*

1/7/00

C-8

Expressions with *ints*

REVIEW:

An integer represents a whole number with no fractional part.

Constants of type *int*:

0, 1, -17, 42 not 0.0 or 1e3

Operators on *ints*:

unary: -

binary: +, -, *, /, %

1/7/00

C-9

int division and remainder

Integer operators include *integer division* and *integer remainder*.

Caution: looks like an old topic, but it's new!

```
  2
100 )299
    200
    ---
     99
```

/ is *integer division*: no remainder, no rounding

299 / 100 → 2, 6 / 4 → 1, 5 / 6 → 0

% is *mod* or *remainder*:

299 % 100 → 99, 6 % 4 → 2, 5 % 6 → 5

1/7/00

C-10

Expressions with *ints*: Time Example

Given: total_minutes 359

Find: hours 5
minutes 59

Solution:

hours = total_minutes / 60 ;

minutes = total_minutes % 60 ;

1/7/00

C-11

A Cautionary Example

int radius;

double area;

▪

▪

▪

*area = (22 / 7) * radius * radius;*

1/7/00

C-12

Why Use *ints*? Why Not *doubles* Always?

- Sometimes only *ints* make sense
 - “give me the 15th spreadsheet cell”
 - “give me the (14.9999998387)th cell” ??
- *Doubles* may be inaccurate representing “ints”
 - In mathematics $3 \cdot 15 \cdot (1/3) = 15$
 - In computer arithmetic $3.0 * 15.0 * (1.0 / 3.0)$ might be 14.99999997
- Last, *and least*
 - arithmetic with *doubles* is slower on some computers
 - *doubles* often require more memory

1/7/00 C-13

Operator Precedence

Precedence determines the order of evaluation of operators.

Is $a + b * a - b$ equal to $(a + b) * (a - b)$ or $a + (b * a) - b$??

And does it matter?

Try this:

$$4 + 3 * 2 - 1$$

$$(4 + 3) * (2 - 1) =$$

$$4 + (3 * 2) - 1 =$$

1/7/00

C-14

Operator Precedence

Precedence rules:

1. do $()$'s first, starting with innermost
2. then do unary minus (negation): -
3. then do “multiplicative” ops: *, /, %
4. lastly do “additive” ops: binary +, -

1/7/00

C-15

Associativity

Associativity determines the order among consecutive operators of equal precedence

Is $a / b * c$ equal to $a / (b * c)$ or $(a / b) * c$??

Most C arithmetic operators are “**left associative**”, **within** the same precedence level

$$a / b * c \text{ equals } (a / b) * c$$

$$a + b - c + d \text{ equals } ((a + b) - c) + d$$

C also has a few operators that are right associative.

1/7/00

C-16

The Full Story...

- C has about 50 operators & 18 precedence levels...
- A “Precedence Table” shows all the operators, their precedence and associativity.
 - Look on inside front cover of our textbook
 - Look in any C reference manual
- When in doubt: check the table
- When faced with an unknown operator: check the table

1/7/00

C-17

Precedence and Associativity: Example

Mathematical formula:

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

C formula:

$$(-b + \text{sqrt}(b * b - 4.0 * a * c)) / (2.0 * a)$$

1/7/00

C-18

Expressions & Values

$b * b - 4.0 * a * c$
 2.5 * 2.5 - 4.0 * -1.0 * 15.2
 6.25 - 4.0 * 15.2
 6.25 - 60.8
 -54.55

1/7/00 C-19

Mixed Type Expressions

What is $2 * 3.14$?

Compiler will implicitly (automatically) convert *int* to *double* when they occur together:

int + *double* → *double* + *double* (likewise -, *, /)

$2 * 3.14 \rightarrow (2 * 3) * 3.14 \rightarrow 6 * 3.14 \rightarrow 6.0 * 3.14 \rightarrow 18.84$
 $2/3 * 3.14 \rightarrow (2/3) * 3.14 \rightarrow 0 * 3.14 \rightarrow 0.0 * 3.14 \rightarrow 0.0$

We **strongly** recommend you avoid mixed types:
e.g., use `2.0 / 3.0 * 3.14` instead.

1/7/00 C-20

Conversions in Assignments

```

int total, count ;
double avg;
total = 97 ; count = 10 ;
avg = total / count ; /*avg is 9.0*/
total = avg; /*BAD!
  
```

1/7/00 C-21

Explicit Conversions

(Section 7.1)

- To be explicit in the program, you can use a **cast**
 - convert the result of an expression to a different type.
- Format: `(type) expression`
- Examples:
 - `(double) myage`
 - `(int) (balance + deposit)`
- This does not change the rules for evaluating the expression (types, etc.)

1/7/00 C-22

Using Casts

```

int total, count ;
double avg;
total = 97 ; count = 10 ;
avg = total / count ; /*avg is 9.0*/
avg = (double) total / (double) count ; /*avg is 9.7*/
avg = (double) (total / count) ; /*avg is 9.0*/
  
```

1/7/00 C-23

C is "Strongly Typed"

- Every variable, value, and expression has a type
- C cares a lot about what the type of each thing is
- Lots of cases where types have to match up
- Start now: be constantly aware of the type of everything in your programs!

1/7/00 C-24

Basic Lessons

- Write in the **clearest** way possible for the reader.
- Keep it **simple**; for very complex expressions, break them up into multiple statements.
- Use **parentheses** to indicate your desired precedence for operators where it may be ambiguous.
- Use explicit **casts** to avoid implicit conversions in mixed mode expressions and assignments.
- Be aware of types.

1/7/00

C-25