CSE 142
Computer Programming I

Arithmetic Expressions

Overview

- Arithmetic expressions
- Integer and floating-point (double) types
- Unary and binary operators
- Precedence
- Associativity
- Conversions and casts
- Symbolic constants

Reading: Text sec. 2.5.

Why Study Expressions?

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

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

Arithmetic on a computer may differ from everyday arithmetic or math:

(1.0 / 9.0) * 9.0 could be 0.99999998213

2 / 3 is zero in C, not .667 (!)

Assignment Statement: Review

```
double area, radius;
area = 3.14 * radius * radius;
```

assignment statement

expression

Execution of an assignment statement:
Evaluate the expression on the right hand side
Store the value of the expression into the variable named on the left hand side

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;

Expression Evaluation

Some terminology:

- Data or operand means the integer or floating-point constants and/or variables in the expression.
- Operators are things like addition, multiplication, etc.

The value of an expression will depend on the data types and values and on the operators used

- Additionally, the final result of an assignment statement will depend on the type of the assignment variable.

Arithmetic Types: Review

C provides two different kinds of numeric values

- Integers (0, 12, -17, 142)
  - Type int
  - Values are exact
  - Constants have no decimal point or exponent
- Floating-point numbers (3.14, -6.023e23)
  - Type double
  - Values are approximate (12-14 digits precision typical)
  - Constants must have decimal point and/or exponent

D-7

Operator Jargon

Binary: operates on two operands

- *3.0 * b*
- *zebra + giraffe*

Unary: operates on one operand

- *-23.4*

C operators are unary or binary

Puzzle: what about expressions like *a+b+c*?

- Answer: this is two binary ops, in sequence

D-8

Expressions with doubles

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: +, -, *, /
- Note: no exponentiation operator in C

D-9

Example Expressions with doubles

Declarations

```
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
```

D-10

Expressions with ints

Constants of type *int*:

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

Operators on *int*s:

- unary: -
- binary: +, -, *, /, %

D-11

int Division and Remainder

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

- symbols / and %

Caution: division looks like an old friend, but there is a new wrinkle!

```
     2  rem 99
100 )299
    200
     99
```

D-12

int Division and Remainder

/ 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

D-13

Expressions with *int*s: Time Example

Given: total_minutes 359

Find: hours 5
minutes 59

Solution in C:

```
hours   = total_minutes / 60 ;
minutes = total_minutes % 60 ;
```

D-14

A Cautionary Example

```
int radius;
double volume;
double pi = 3.141596;
.
.
volume  = ( 4/3 ) * pi * radius  *radius * radius;
```

D-15

Why Use *int*s? Why Not *double*s Always?

Sometimes only *int*s make sense

- the 15th spreadsheet cell, not the 14.997th cell

*Double*s may be inaccurate representing "ints"

- In mathematics 3 • 15 • (1/3) = 15
- But, 3.0 * 15.0 * (1.0 / 3.0) might be 14.9999997

Last, *and least*

- operations with *double*s is slower on some computers
- *double*s often require more memory

D-16

Order of Evaluation

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) = 7

4 + (3 * 2) - 1 = 9

D-17

Operator Precedence Rules

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 +, -

D-18

Precedence Isn't Enough

Precedence doesn't help if all the operators have the same precedence

*Is a / b * c* equal to

*a / (b * c)* or *(a / b) * c* ??

Associativity determines the order among consecutive operators of equal precedence

Does it matter? Try this: 15 / 4 * 2

D-19

Associativity Matters

Associativity determines the order among consecutive operators of equal precedence

Does it matter? Try this 15 / 4 * 2

(15 / 4) * 2 = 3 * 2 = 6

15 / (4 * 2) = 15 / 8 = 1

D-20

Associativity Rules

Most C arithmetic operators are "left associative", within the same precedence level

*a / b * c* equals *(a / b) * c*

a + b - c + d equals *((a + b) - c) + d*

C also has a few operators that are right associative

D-21

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

D-22

Precedence and Associativity: Example

Mathematical formula:

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

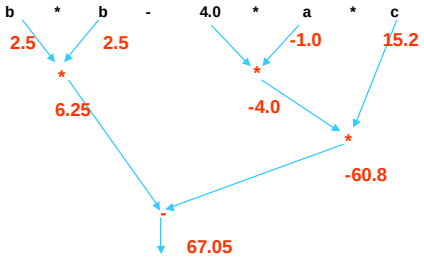
C formula:

*(- b + sqrt (b * b - 4.0 * a * c)) / (2.0 * a)*

D-23

Depicting Expressions

b = 2.5;
a = -1.0;
c = 15.2;



D-24

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 * 3.14 → (2*3) * 3.14 → 6 * 3.14 → 6.0 * 3.14 → 18.84

2/3 * 3.14 → (2/3) * 3.14 → 0 * 3.14 → 0.0 * 3.14 → 0.0

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

Conversions in Assignments

```
int total, count, value;
double avg;
total = 97 ;    count = 10;
avg = total / count;    /*avg is 9.0!*/
value = total*2.2;      /*bad news*/
```

implicit conversion to double

implicit conversion to int – drops fraction with no warning

D-26

Explicit Conversions

Use a cast to explicitly 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 itself (types, etc.)

Good style, because it shows the reader that the conversion was intentional, not an accident

D-27

Using Casts

```
int total, count ;
double avg;
total = 97 ;    count = 10 ;

   /* explicit conversion to double (right way)*/
avg = (double) total / (double) count;    /*avg is 9.7 */
   /* explicit conversion to double (wrong way)*/
avg = (double) (total / count) ;          /*avg is 9.0*/
```

D-28

#define - Symbolic Constants

Named constants:

```
#define PI          3.14159265
   ...
circle_area = PI * radius * radius ;
```

Note: = and ; are not used for #define

D-29

Expressions in #define

```
#define PI          3.14159265
#define HEIGHT      50
#define  WIDTH      80
#define AREA        (HEIGHT * WIDTH)
   ...
circle_area = PI * radius * radius ;
volume = length * AREA;
```

() can be used in #define

() should be used for any non-simple expression

D-30

Why #define?

Centralize changes

No "magic numbers" (unexplained constants)

use good names instead

Avoid typing errors

Avoid accidental assignments to constants

```
double pi ;    vs.
pi = 3.14 ;          #define PI 3.14
...                  ...
pi = 17.2 ;          PI = 17.2 ;      syntax error
```

D-31

Types are Important

- *Every* variable, value, and expression in C has a type
- Types matter - they control how things behave (results of expressions, etc.)
- Lots of cases where types have to match up
- Start now: be constantly aware of the type of everything in your programs!

D-32

Advice on Writing Expressions

- Write in the clearest way possible to help the reader
- Keep it simple; break very complex expressions into multiple assignment statements
- Use parentheses to indicate your desired precedence for operators when it is not clear
- Use explicit casts to avoid (hidden) implicit conversions in mixed mode expressions and assignments
- Be aware of types

D-33

Next Time

We'll discuss input and output

See you then!

D-34