(2-1) Numeric Expressions in C H&K Chapter 2

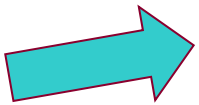
Instructor – Beiyu Lin
CptS 121 (May 8th, 2019)
Washington State University

Review --- General Form in C

```
#include<stdio.h>    /* starting with including libraries*/
#include<stdlib.h>

int main(void)
{
      Instructions for the machine to execute

      return 0;
}
```



Instructions include:

1. Declare a variable / user-defined identifier:
e.g. int height = 0, volume = 0;
e.g. char char_variable;
e.g. double radius = 0.0;

2. Programming assignment:
e.g. volume = ((double) 1/3) *height*radius*radius;

3. Input and output statements:
e.g. printf("the calculated volume is %f", volume);
e.g. scanf("%c", &char_variable);

Review – Possible Errors

- Rarely will you write a program that is free of errors
- You'll need to diagnose and correct three kinds of errors:
  - Syntax errors (code violates syntax rules for a proper C program)
    - Detected at compile time
    - An executable file will not be generated unless they're corrected
    - Examples:
      - Missing semi-colon
      - Unmatched brace
      - Undeclared identifiers
      - Failure to close a comment properly
    - Note: Removing one error may make others disappear (the compiler gets confused easily) – Always start with the first error listed by the compiler!

Review – Possible Errors

- Rarely will you write a program that is free of errors
- You'll need to diagnose and correct three kinds of errors:
  - Syntax errors (code violates syntax rules for a proper C program)
    - Detected at compile time
    - An executable file will not be generated unless they're corrected
    - Examples:
      - Missing semi-colon
      - Unmatched brace
      - Undeclared identifiers
      - Failure to close a comment properly
    - Note: Removing one error may make others disappear (the compiler gets confused easily) – Always start with the first error listed by the compiler!

Programming Errors (2)

- Run-time errors
  - Commonly called "bugs"
  - Cause the program to "crash": an error is reported, and control is turned over to the operating system
  - Examples
    - Division by zero
    - Referencing a memory cell that's out of range
    - Getting into an infinite loop, which may ultimately cause a "stack overflow"
    - Referencing a null pointer (more on this later…)

Arithmetic Expressions

- Most programming problems require arithmetic expressions as part of solution; including problems related to:
  - Mechanics
  - Kinematics
  - Materials science
  - Electronics
  - Many others…
- Require numerical operands
- Form: operand1 operator operand2
- Type of result dependent on operand types

Arithmetic Operators in C (1)

Operator	Representation	Example
+	Addition	10 + 5 = 15 1.55 + 13.3 = 14.85 3 + 100.7 = 103.7
-	Subtraction	10 – 5 = 5 5.0 – 10.0 = -5.0 10 – 5.0 = 5.0
*	Multiplication	1 * 5 = 5 1.000 * 10.0 = 10.0 5 * 5.0 = 25.0

Arithmetic Operators in C (2)

Operator	Representation	Example
/	Division	2 / 3 = 0 10.0 / 4.0 = 2.5 10 / 3.0 = 3.3333
%	Modulus	5 % 2 = 1 2 % 5 = 2 6 % 0 = undefined 6.0 % 3 = won't compile

Mixed-Type Expressions

- Types of operands in expression are different
  - An integer value and a double value
- The result is always the more precise data type
  - 10 (an int) + 25.5 (a double) = 35.5 (a double)

Mixed-Type Assignment Statements

- Evaluated from right-to-left
- Expression is first evaluated (what's on right-hand-side) and then assigned to variable (what's on left-hand-side)
- Examples:

```
int result_int, op1_int = 5, op2_int = 42;
double result_double, op1_double = 5.5;
result_int = op1_int + op1_double; /* mixed expression, integer
    assignment, result_int = 10 (truncation occurs) */
result_double = op1_int + op2_int; /* integer expression, double
    assignment, result_double = 47.0*/
result_double = op1_int + op1_double; /* mixed expression, double
    assignment, result_double = 10.5*/
```


Type Conversions & Type Casts

- Changing one entity of a data type into another
- Two kinds exist:
  - Implicit
  - Explicit
- Implicit type conversion example:

```
int num1 = 12;
double num2;
num2 = num1; /* num1 implicitly casted to type double, 12.0 */
```

- Explicit type conversion example:

```
double num1;
num1 = ((double) 1 / 5); /* integer 1 explicitly casted to type double,
    1.0 */
```


Multiple Operator Expressions

- May contain unary and binary operators
- Unary operators consists of one operand
- Binary operators require two operands
- Example:

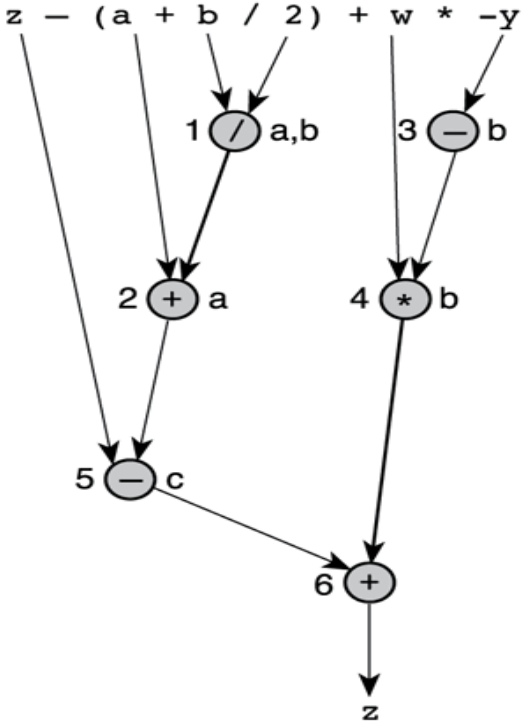
```
y = -x + x * x / 10; /* -x applies the unary sign
    operator for negation */
```

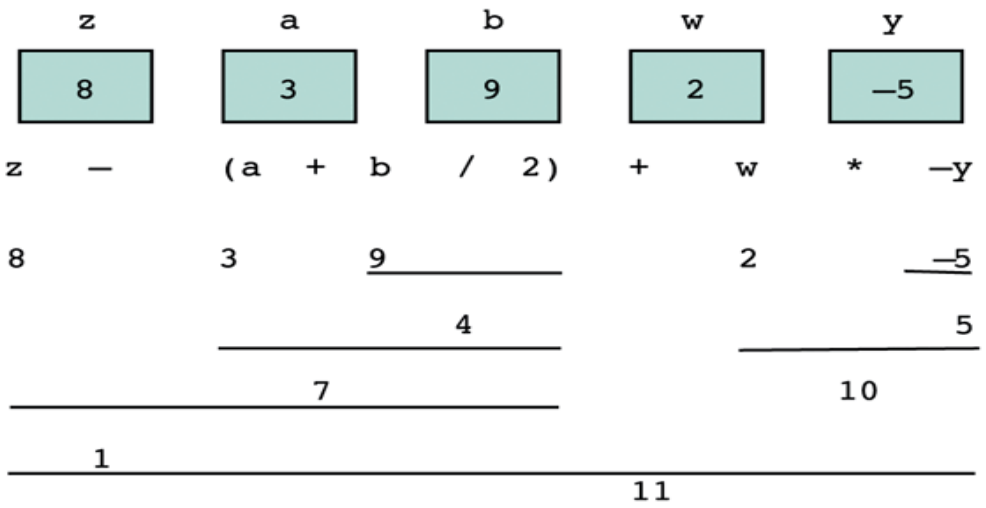

Operator Precedence (1)

- Operator Precedence
  - How is x – y / z evaluated?
    - (x – y) / z ?
    - x – (y / z) ?
  - Important to understand operator precedence rules:
    - Evaluation proceeds left to right
    - Subexpressions in parentheses are evaluated first
    - In cases where no parentheses are used, *, /, and % take precedence over + and –
  - So x – y / z is evaluated as x – (y / z), because / takes precedence over –
  - Note: The *unary* operators + and – are used to indicate the sign of a number (e.g., +5, -3.0). They take precedence over all binary operators, and are evaluated right to left:
    - Example: -3 + 5 * 4 would be evaluated as (-3) + (5 * 4) = 17.
  - *Recommendation*: Liberally use parentheses to avoid confusion and unexpected results!

Operator Precedence (2)

- Operator Precedence Example (H & K p. 80)





Formatting Numbers (1)

- C defines "default" output style for each data type
  - No leading blanks for int and double
  - double displayed with default number of digits to right of decimal point (how many?)
- You can override these defaults by specifying custom format strings to `printf` function

```
int x;
double y;
x = 3;
y = 2.17;
printf("x is %3d. y is %5.1f.",x,y);
```

Output:

```
x is  3. y is  2.2.
```


Formatting Numbers (2)

- Notes:
  - For double output, format string is of form `%n.mf`, where `n` is *total width* (number of columns) of formatted number, and `m` is the number of digits to the right of decimal point to display.
  - It is possible to omit `n`. In that case, no leading spaces are printed. `m` can still specify the number of decimal places (e.g., `%.2f`)

Formatting Numbers (3)

- You try it:
  - If the values of the variables `a`, `b`, and `c` are 504, 302.558, and -12.31, write a statement that will display the following line (☐is used to denote a blank):

```
☐☐504☐☐☐☐☐302.56☐☐☐☐-12.3
```

```
printf("%5d%11.2f%9.1f",a,b,c);
```


Programming Errors (3)

- Logic Errors
  - Cause the program to compute incorrect results
  - Often go unnoticed, at least at first
  - Examples
    - Your algorithm is wrong because you misunderstand the problem
    - You do not obtain input data properly, so your computations work on the wrong data. For example:

```
int year;
char first, middle, last;
printf("Enter the current year and press return: ");
scanf("%d", &year);
printf("Type in 3 initials and press return: ");
scanf("%c%c%c", &first, &middle, &last);
```

What goes wrong here?

Next Lecture…

- Top-Down Design and Functions

References

- J.R. Hanly & E.B. Koffman, *Problem Solving and Program Design in C (8th Ed.)*, Addison-Wesley, 2016

Collaborators

- Chris Hundhausen
- A. O'Fallon