Excerpts from Chapter 3: Expressions and Interactivity

slides from Gaddis, Walters & Muganda (2017). Starting Out with C++ Early Objects 9th Ed.

Topics

3.1 The `cin` Object

- `cin` is the standard input object
- Like `cout`, requires `iostream` header
- Used to read input from keyboard
- Often used with `cout` to display a user prompt first
- Data is retrieved from `cin` with **>>**, the stream extraction operator
- Input data is stored in one or more variables

The **`cin`** Object

- User input goes from keyboard to the input buffer, where it is stored as characters
- **`cin`** converts the data to the type that matches the variable

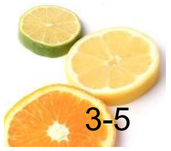
```
int height;
std::cout << "How tall is the room? ";
std::cin  >> height;
```

The `cin` Object

- Can be used to input multiple values

```
std::cin >> height >> width;
```

- Multiple values from keyboard must be separated by spaces or [Enter]
- Must press [Enter] after typing last value
- Multiple values need not all be of the same type
- Order is important; first value entered is stored in first variable, etc.



3.2 Mathematical Expressions

- An expression is something that can be evaluated to produce a value.
- It can be a constant, a variable, or a combination of constants and variables combined with operators and grouping symbols
- We can create complex expressions using multiple mathematical operators
- Examples of mathematical expressions:

```
 2
height
a + b / c
```

Using Mathematical Expressions

- Can be used in assignment statements, with **`cout`**, and in other types of statements
- Examples:

```
area = 2 * PI * radius;
std::cout << "border is: " << (2*(l+w));
```

This is an expression

These are expressions

Order of Operations

In an expression with > 1 operator, evaluate it in this order:

Do first: ( ) expressions in parentheses

Do next: `-` (unary negation) in order, left to right

Do next: `*  /  %` in order, left to right

Do last: `+  -` in order, left to right

Ex: In the expression `2 + 2 * 2 - 2` ,

Evaluate 2nd (`+`)   **Evaluate 1st** (`*`)   **Evaluate 3rd** (`-`)

Algebraic Expressions

- Multiplication requires an operator

  $Area = lw$ is written as `Area = l * w;`

- There is no exponentiation operator

  $Area = s^2$ is written as `Area = pow(s, 2);`

  (note: `pow` requires the `cmath` header file)

- Parentheses may be needed to maintain order of operations

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

is written as

```
m = (y2-y1)/(x2-x1);
```

3.3 Data Type Conversion and Type Casting

- Operations are performed between operands of the same type
- If operands do not have the same type, C++ will automatically convert one to be the type of the other
- This can impact the results of calculations

Hierarchy of Data Types

- Highest `long double`

```
double
float
unsigned long long int
long long int
unsigned long int
long int
unsigned int
```

- Lowest `int`
- Ranked by largest number they can hold

Type Coercion

- Coercion: automatic conversion of an operand to another data type
  - Promotion: conversion to a higher type
  - Demotion: conversion to a lower type

Coercion Rules (Promotion)

1) **`char`**, **`short`**, **`unsigned short`** are automatically promoted to **`int`**
2) When operating with values of different data types, the lower-ranked one is promoted to the type of the higher one.
3) When using the = operator, the type of expression on right will be converted to the type of variable on left

Coercion Rules – Important Notes

1) If demotion is required by the = operator,
   - the stored result may be incorrect if there is not enough space available in the receiving variable
   - floating-point values are truncated when assigned to integer variables

2) Coercion affects the <u>value</u> used in a calculation. It does not change the type associated with a variable.

3.5 Named Constants

- Also called constant variables
- Variables whose content cannot be changed during program execution
- Used for representing constant values with descriptive names

```
const double TAX_RATE = 0.0775;
const int NUM_STATES = 50;
```

- Often named in uppercase letters

Defining and Initializing Named Constants

- The value of a named constant must be assigned when the variable is defined:

```
const int CLASS_SIZE = 24;
```

- An error occurs if you try to change the value stored in a named constant after it is defined:

```
// This won't work
CLASS_SIZE = CLASS_SIZE + 1;
```

Benefits of Named Constants

- They make program code more readable by documenting the purpose of the constant in the name:

```
const double TAX_RATE = 0.0775;
. . .
sales_tax = purchase_price * TAX_RATE;
```

- They improve accuracy and simplify program maintenance:

3.6 Multiple and Combined Assignment

- The assignment operator (=) can be used multiple times in an expression

```
x = y = z = 5;
```

- Associates right to left

```
x = (y = (z = 5));
```

Done 3rd   Done 2nd   Done 1st

Combined Assignment

- Applies an arithmetic operation to a variable and assigns the result as the new value of that variable
- Operators: `+=` `-=` `*=` `/=` `%=`
- These are also called compound operators or arithmetic assignment operators
- Example:

`sum += amt;` is short for `sum = sum + amt;`

More Examples

```
x += 5;   means   x = x + 5;
x -= 5;   means   x = x – 5;
x *= 5;   means   x = x * 5;
x /= 5;   means   x = x / 5;
x %= 5;   means   x = x % 5;
```

The right hand side is evaluated before the combined assignment operation is done.

```
x *= a + b;   means   x = x * (a + b);
```

Excerpts from Chapter 3: Expressions and Interactivity

slides from Gaddis, Walters & Muganda (2017). Starting Out with C++ Early Objects 9th Ed.