Operators

Class 5

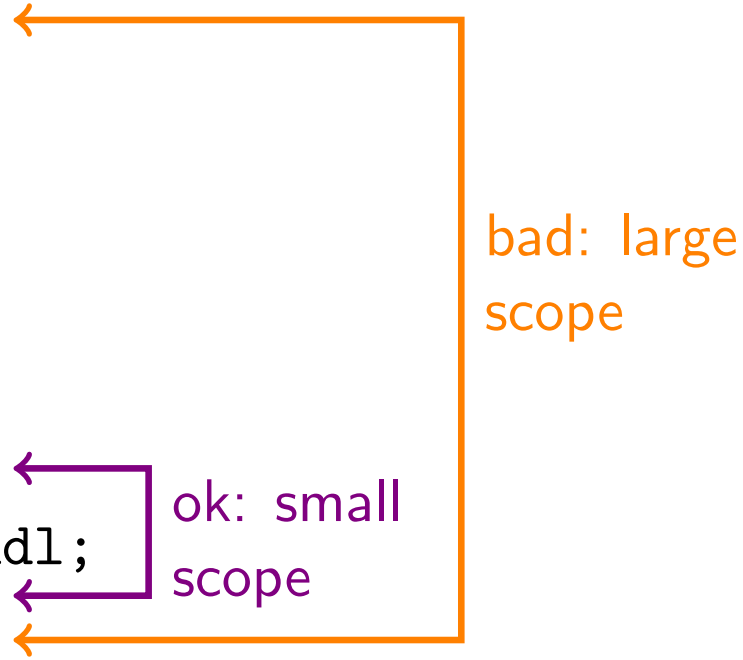
Section 2.11

- we will not cover section 2.11
- you will not be tested on it
- feel free to read it on your own

Declaration Location

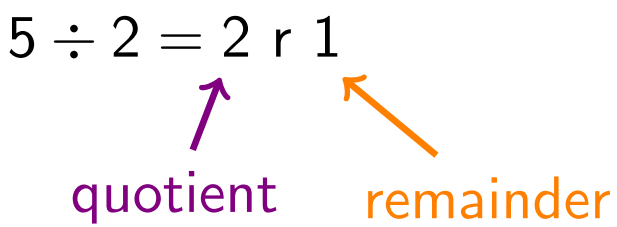
- best practice in coding dictates that in general a variable's scope should be as small as possible
- this means declaring a variable as close to the place where it is used as possible
- the style guide also has this rule

```
double rate;
double hours;
cout << "Enter hours: ";
cin >> hours;
cout << "Enter rate: ";
cin >> rate;
double pay = hours * rate;
cout << "Pay: " << pay << endl;
return 0;
```



Modulus

- applies only to integer types
- not defined for floating point types (Python allows this, which is just weird)
- used to give the remainder after division completes



Modulus

- when using modulus, the dividend can be positive, zero, or negative
- the divisor should always be positive (just like in elementary school)
- a negative divisor is legal in C++, but mathematically very controversial, so don't do it

Modulus

- an extremely useful operator
- two big uses:
  1. determine if a number is even or odd
  2. (combined with division) determine specific digits in a base-10 number

Even or Odd

- a number is even if its remainder when divided by 2 is 0
- 156 % 2 is 0, so 156 is an even number

- a number is odd if its remainder when divided by 2 is 1
- 157 % 2 is 1, so 157 is an odd number

Digits in a Number

- imagine I have: `int x = 12345;`
- I need to extract the hundreds digit (in this case 3) from the value
- `hundreds_digit = (x / 100) % 10;`

Digits in a Number

- imagine I have: `int x = 12345;`
- I need to extract the hundreds digit (in this case 3) from the value
- `hundreds_digit = (x / 100) % 10;`
- how would you get the thousands digit?

Digits in a Number

- imagine I have: `int x = 12345;`
- I need to extract the hundreds digit (in this case 3) from the value
- `hundreds digit = (x / 100) % 10;`
- how would you get the thousands digit?

- an example use: convert 255 minutes into minutes and hours

Digits in a Number

- imagine I have: `int x = 12345;`
- I need to extract the hundreds digit (in this case 3) from the value
- `hundreds digit = (x / 100) % 10;`
- how would you get the thousands digit?

- an example use: convert 255 minutes into minutes and hours

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

unsigned check = hours * 60 + minutes;
```

Parentheses

- just like in algebra, we use parentheses to override precedence
- `x = 1 + 2 * 3;` multiplication has higher precedence than addition, so this yields 7
- `x = (1 + 2) * 3;` parentheses override precedence, so this yields 9

Parentheses

- just like in algebra, we use parentheses to override precedence
- `x = 1 + 2 * 3;` multiplication has higher precedence than addition, so this yields 7
- `x = (1 + 2) * 3;` parentheses override precedence, so this yields 9

- however, only use parentheses either 1) when they are necessary or 2) when they improve readability
- `x = 1 * (2 * 3);` bad: irrelevant because multiplication is commutative
- `x = (1 + 2 + 3);` bad: unnecessary, and thus confusing
- `x = (1 + 2) / (3 + 4);` good: mathematically necessary
- `x = (2 * a) + (b * 4 / c);` good: not mathematically necessary, but improves readability

Magic Numbers

- in programming, an anonymous value is called a magic number
- it is a literal that appears in code with no hint of its purpose
- 0.069 is a magic number
- if instead we wrote:

  ```
  amount = balance * interest rate;
  ```

  it would be obvious what was going on
-

Repeated Use

- in a banking program, there may be many places where the interest rate is involved in calculations
- if 0.069 is used 37 times over 25 pages of code, two bad things can happen
  1. what if the interest rate changes to, say, 0.066? all 37 occurrences of 0.069 over 25 pages must be found and changed!
  2. even if the rate doesn't change, what if one of the 37 occurrences is mis-typed as 0.068? the chances of noticing it are slim

Repeated Use

- in a banking program, there may be many places where the interest rate is involved in calculations
- if 0.069 is used 37 times over 25 pages of code, two bad things can happen
  1. what if the interest rate changes to, say, 0.066? all 37 occurrences of 0.069 over 25 pages must be found and changed!
  2. even if the rate doesn't change, what if one of the 37 occurrences is mis-typed as 0.068? the chances of noticing it are slim

- to avoid both anonymous magic numbers and repeated use of a literal, we use named constants

Named Constants

```
const double INTEREST_RATE = 0.069;

pay = sum * INTEREST_RATE;
```

- if the interest rate changes, you need only change the code in one place
- interest rate is always identical; no chance of a typo
- since the interest rate never changes in one run of the program, it is a constant
- like a variable, but cannot be re-assigned
- constant identifiers should be in ALL UPPER CASE

From Lab 2

- some things noticed in lab 2 submissions

```
double diameter;
diameter = radius * 2.0;
```

```
double diameter = radius * 2.0;
```

Which is preferable, and why?

From Lab 2

- some things noticed in lab 2 submissions

```
double diameter;
diameter = radius * 2.0;
```

```
double diameter = radius * 2.0;
```

Which is preferable, and why?

The right side, because it's more concise. It conveys the exact same meaning with fewer words, and is thus more readable.

From Lab 2

- some things noticed in lab 2 submissions

```
double diameter;
diameter = radius * 2.0;
```

```
double diameter = radius * 2.0;
```

Which is preferable, and why?

The right side, because it's more concise. It conveys the exact same meaning with fewer words, and is thus more readable.

`diameter`	`circle_diameter`
`radius`	`circle_radius`
`area`	`circle_area`

Which is preferable, and why?

From Lab 2

- some things noticed in lab 2 submissions

```
double diameter;
diameter = radius * 2.0;
```

```
double diameter = radius * 2.0;
```

Which is preferable, and why?

The right side, because it's more concise. It conveys the exact same meaning with fewer words, and is thus more readable.

```
diameter
radius
area
```

```
circle diameter
circle radius
circle area
```

Which is preferable, and why?

The left side, because this program only involves a circle. If this were a general-purpose geometry program, then triangle_area vs circle_area might be essential. But here, circle_ is simply noise.

From Lab 2

```
const double PI_CONSTANT = 3.1415;
```

or

```
const double MATHEMATICAL_PIE = 3.1415;
```

or

```
const double PI = 3.1415;
```

Which one is preferable, and why?

From Lab 2

From the assignment: " ... the program should produce output that looks exactly like this:"

```
Please enter the radius of a circle: 14
For a circle with radius 14
The diameter is 28
The circumference is 87.962
The area is 615.734
```

From a student's program:

```
This program Calculates the properties of a circle
given its radius
Please enter the circle's radius: 14
The diameter is 28 units
The circumference is 87.962 units
The area is 615.734 units
```

If the specifications call for a particular behavior, you the developer must meet those specifications.

The cin Object

- cout is an output stream object connected to the screen
- cin is an input stream object connected to the keyboard
- cout is used with the stream insertion operator <<
- cin is used with the stream extraction operator >>

- the stream extraction operator extracts a value from the stream of characters coming from the keyboard and assign the value to the variable

The cin Object

- the variable that is used with cin must have been declared prior to use
- you cannot initialize a variable with cin

OK:

```
double radius;
cin >> radius;
```

Illegal:

```
cin >> double radius;
```

cin with Multiple Values

- cin extraction may be used to gather multiple values in one statement
- see program 3-2 on page 87:

```
unsigned length;
unsigned width;
cout << "Enter the length and width: ";
cin >> length >> width;
unsigned area = length * width;
```

Multiple Values of Varying Types

- cin can extract multiple values of different types from the input stream

```
unsigned count;
double measurement;
cout << "Enter the count and the measurement: ";
cin >> count >> measurement;
```

Whitespace

- whitespace is the term used to describe any sequence of one or more characters that the eye perceives as a separation of words
- the main whitespace characters are space, tab, and newline

Keyboard Buffer

- in computer science, a buffer is an area of memory where characters are stored while waiting to be processed
- In a Console program as we type on a keyboard, the keystrokes are stored in the keyboard buffer
- the Console based program only starts reading the characters when the user presses the Enter key
- all characters before the Enter key are stored in the keyboard buffer
- you can use backspace and change the input characters as long as the Enter key has not been pressed
- when the Enter key is pressed, all characters plus the Enter key are sent to the Console program

Delimiting cin Input

- cin extraction skips whitespace before starting to extract a value

- cin extraction is greedy as it reads the keyboard stream
- it tries to read as many characters as possible
- it stops reading only when it can't go farther

- cin extraction throws away everything in the buffer between the last thing it can read and the Enter key

run program try_cin with various inputs, correct and incorrect:

cin Notes

- cin is very easy to use, BUT:
  - it is impossible to read the space character into a char variable
  - cin extraction cannot read a string with an embedded space
  - cin extraction does no error checking
  -
- we will later see better and safer ways of getting input

Combined Assignment

- the most common statement in programming is assignment:
  `foo = bar;`
- the second most common statement pattern is an arithmetic operation on a variable followed by assignment to that variable:
  `foo = foo + 3;`
- this pattern is very common in C++
- it involves the name of the variable typed twice
- C++ has a shortcut form that combines arithmetic and assignment in one symbol

```
foo = foo + 3;
foo += 3;
```