09. Input Validation, Arithmetic Operators, Assignment Operators

CPSC 120: Introduction to Programming
Kevin A. Wortman ~ CSU Fullerton

Agenda

0. Sign-in sheet
1. Technical Q&A
2. Input Validation
3. Arithmetic Operators
4. Assignment Operators

1. Technical Q&A

Technical Q&A

Let’s hear your noted questions about...

- This week’s Lab
- Linux
- Any other technical issues

Reminder: write these questions in your notebook during lab

3. Assignment Operators

Code Variables Can Change

- In math, variables do not change, so
  x = 5
  x = 6
  are an invalid contradiction

- In programming, **variables can change over time**
- So this is fine:
  int x { 5 };
  x = 6; // change x to store 6

Side Effects

- Some expressions have side effects
- **Side effect:** a change that is a consequence of evaluating an expression

Kind of Operator	Side Effect	Example
Arithmetic	None	`this_year - birth_year`
stream insertion <<	Print value	`std::cout << year`
stream extraction >>	Store input in variable	`std::cin >> year`
Assignment (next slide)	Store expression in variable	`age = this_year - birth_year`

Assignment Operator

expression:

left **=** *expr*

Semantics:

- *left* must be a variable (or other *lvalue*)
- Evaluate *expr* to produce an object
- Side effect: *left* now stores the new object

left is **changed**
(unlike = in math)

Examples:

```
int score{ 0 };
std::cout << score << “\n”; // prints 0
score = 5;
std::cout << score << “\n”; // prints 5
score = -1;
std::cout << score << “\n”; //prints -1
```

Pitfall: Backwards Assignment

- Pattern:

  left **=** *expr*

- **Changes *left*** to become *expr*
- Pitfalls:
  - Expression on left side
  - Destination on left side

```
int a{ 3 }, b { 9 };
4 = a;  // compile error

// intend to change a to hold b's value
b = a; // backwards, should be a = b;
```

Review: Expression Statement

statement:

expr ;

Example:

```
std::cout << “Hi” << “ there”;
```

Semantics:

- Evaluate *expr*
- Any object produced by *expr* is discarded
- (That’s all)

Pitfall: Ineffectual Expression Statement

- **Ineffectual**: has no effect
- Recall: object produced in an expression statement is discarded
- An expression statement with no side effects is ineffectual
  - Accomplishes nothing
  - Programmer may be confused
  - Delete the statement

Example:

```
int score { 0 };
score + 1;  // ineffectual
std::cout << score << “\n”;  //prints 0
```

Programmer intended:

```
int score { 0 };
score = score + 1;
std::cout << score << “\n”;  //prints 1
```

Arithmetic Assignment Operators

- Pattern: assign a variable to a new version of itself
- **Arithmetic assignment operator:** combination of = and an arithmetic operator
- Syntax:

*left op***=** *expr*

- Semantics: equivalent to

left = left op expr

Arithmetic Assignment	Equivalent To
`count += 1;`	`count = count + 1;`
`radius *= s;`	`radius = radius * s;`
`width /= 2;`	`width = width / 2;`
`roll %= sides;`	`roll = roll % sides;`

Pre-Increment And Pre-Decrement

- **Increment**: increase by one
- **Decrement**: decrease by one
- Common operation (ex. counting things)

Operator	Semantics	Example
`++`*`var`*	increment *var*	`++count;`
`--`*`var`*	decrement *var*	`--lives;`

Post-Increment Operators

- **Post-increment**: increments *var* **after producing the original value**
- **Post-decrement**: decrements *var* **after producing the original value**
- write operator **after** *var*

Operator	Semantics	Example
var`++`	produce current value of *var*, and then increment *var*	`count++;`
var`--`	produce current value of *var*, and then decrement *var*	`lives--;`

- Easter egg: C++ “one-ups” C

Example: Increment

```
#include <iostream>
int main(int argc, char* argv[]) {

  int a{ 5 };
  std::cout << "a is " << a << “\n”;
  a++;
  std::cout << "a is " << a << “\n”;
  std::cout << "a is " << ++a << “\n”;
  std::cout << "a is " << a << “\n”;
  std::cout << "a is " << a++ << “\n”;
  std::cout << "a is " << a << “\n”;

  return 0;
}
```

```
$ ./a.out
a is 5
a is 6
a is 7
a is 7
a is 7
a is 8
```