

Ch 3: Expressions and Interactivity

Part I

CS 1428

Fall 2011

Jill Seaman

Lecture 5

1

Console Input: cin

- Used to get input from the user.
- **cin**: console input (from the keyboard)
 - a stream object: works on a sequence of data
- **>>**: the stream extraction operator
 - Extracts value from stream (lhs) and stores in variable on right-hand side (rhs)
 - `cin >> myVariable;`
 - skips over white-space (space, newline) to get the next value.
 - Automatically converts characters typed by the user to the type of the variable on the rhs.
 - This statement waits for the user to type a value.

2

Console Input: cin

- Output a prompt (using cout) to tell the user what type of data to enter BEFORE using cin.

```
int diameter;  
  
cout << "What is the diameter of the circle? ";  
cin >> diameter;
```

- Waits for user to enter a number followed by enter/newline.
- Make sure arrows point in the right direction
 - output: to stream
 - input: to variable

3

Console Input: Multiple Values

- You can input multiple values in one line:

```
int x, y;  
  
cout << "Enter two integers: " << endl;  
cin >> x >> y;
```

- The user may enter them either
 - on one line, separated by space
 - on separate lines
- The user must enter values of the expected data type.

4

(Mathematical) Expressions

- An **expression** is a program component that evaluates to a **value**.
- Examples:

<code>x + 5</code>	<code>x * y / z</code>
<code>num</code>	<code>'A'</code>
<code>4</code>	<code>-15e10</code>
<code>8 * x * x - 16 * x + 3</code>	

- Each expression has a type, which is the type of the result value.

5

Where can expressions occur?

- The rhs of an assignment statement:

```
x = y * 10 / 3;  
y = 8;  
num = num + 1;  
aLetter = 'W';  
x = y;
```

- The rhs of a stream insertion operator (<<):

```
cout << "The pay for the week is " << hours * rate << endl;  
cout << num;  
cout << 25 / y;
```

6

Operator Precedence

- Which operation gets done first?

```
answer = 1 + x + z;  
result = x + 5 * y;
```

- Precedence Rules: Higher up done first
- Associativity: operators on same level are performed either left to right or right to left:

- - (unary minus)	Right to left
- * / %	Left to right
- + -	Left to right

- 5 + 2 * 4
160 / 4 * 2
4 + 17 % 2 - 1

7

Parentheses

- You can use parentheses to override the precedence or associativity rules.

```
a + b / 4  
(a + b) / 4  
(4 * 17) + (3 - 1)  
a - (b - c)
```

- Run the expressions.cpp demo with input values: 30 20 5

8

Exponents

- There is no operator for exponentiation in C++
- There IS a library function called “pow”

```
result = pow(x, 3.0); // x cubed, or x to the third power
```

- The expression is a call to the pow function with arguments x and 3.0.
- Arguments should have type double and the result is a double.
- If x is 2.0, the result is 8.0.
- `#include <cmath>` is required to use pow.

9

Type Conversion

- Implicit type conversion (type coercion) occurs when an expression has an unexpected type.
- The compiler converts the expression to the desired type automatically.
- Expressions of lower-ranking type are converted to higher-ranking type.
 - double
 - float
 - long
 - int
 - char

10

Type Conversion Rules

- Binary operations convert lower ranking value to the type of the other expression/value.

```
int years;  
float interestRate;  
result = years * interestRate;
```

// years is converted to float before being multiplied

- The rhs of assignment operator is converted to the type of the variable on the lhs.

```
int x, y = 4;  
float z = 2.7;  
x = y * z;
```

// y is converted to float, 10.8 is converted to int (10) 11

Integer Division

- When an integer is divided by an integer the result is an integer.
- The remainder/fractional part is discarded, NO ROUNDING.

```
double result;  
result = 15 / 6;      // 2.5 ==> 2 ==> 2.0  
result = 15.0 / 6;    // 6 ==> 6.0, result is 2.5
```

Type Casting

- Type casting is an explicit or manual type conversion.
- `static_cast<datatype>(expr)`
- mainly used to force floating-point division

```
int hits, atBats;  
float battingAvg;  
...  
cin >> hits >> atBats;  
battingAvg = static_cast<float>(hits) / atBats;
```

- why not: `static_cast<float>(hits / atBats)`
?

13

Overflow/Underflow

- When the value assigned to a variable is too large or small for its type.
- integers tend to wrap around, without warning:

```
short testVar = 32767;  
cout << testVar << endl; // 32767, max value  
testVar = testVar + 1;  
cout << testVar << endl; //-32768, min value
```

- floating point value overflow/underflow:
 - may or may not get a warning
 - result may be 0 or random value

14