Relational Operators and if

Class 10

Data Type

- a data type consists of two things:

Data Type

- a data type consists of two things:
  - a set of values
  - a set of operations defined on those values

Data Type

- a data type consists of two things:
  - a set of values
  - a set of operations defined on those values
- the numeric types have the operations
  - unary negation
  - binary addition and subtraction
  - multiplication, division (floating and integer), modulus (integer)
  - assignment $=$ $*=$ $+=$ and several more

Relops

- there is another whole family of operators for the numeric types
- they are the relational operators, or relops for short
- they allow us to compare one value to another
- the relops are given in table 4-1 on page 152, and are:

Operator	Meaning
$>$	greater than
$<$	less then
$>=$	greater than or equal to
$<=$	less than or equal to
$==$	equal to
$!=$	not equal to

Relops and Types

- just like arithmetic operators, relops have rules about mixing types
- you can compare
  - integer types of the same size and kind (signed or unsigned)
  - integer types of different sizes but the same kind
  - floating point types of the same size using the four lt and gt relops only
  - a floating point type and an integer type using the four lt and gt relops only if you really must mix types
- you cannot compare
  - floating point types using the two equality relops == and !=
  - floating point types of different sizes
  - signed and unsigned integer types

Return Type

- when you add two ints, you get an int
  `int + int` $\rightarrow$ `int`
- when you compare two ints, you get a bool
  `int < int` $\rightarrow$ `bool`

Return Type

- when you add two ints, you get an int
  `int + int` $\rightarrow$ `int`
- when you compare two ints, you get a bool
  `int < int` $\rightarrow$ `bool`

- in other words, the returned value as a result of comparing two ints is a Boolean value
- the statement 5 < 10 is an assertion
- it makes the claim, "5 is less than 10"
- the claim is either true or false
- thus, the operator's result type is Boolean

Assertions

- the relops are used to make assertions that evaluate to either true or false
  - `5 < 10` is a true assertion
  - `10 < 5` is a false assertion

- an expression formed using a relop is a Boolean expression
- a Boolean expression is either true or false

- let `x = 10;` and `y = 7;`

Expression	**Value**
`x < y`	
`x > y`	
`x >= y`	
`x == y`	
`x != y`	

Assertions

- the relops are used to make assertions that evaluate to either true or false
  - `5 < 10` is a true assertion
  - `10 < 5` is a false assertion

- an expression formed using a relop is a Boolean expression
- a Boolean expression is either true or false

- let `x = 10;` and `y = 7;`

Expression	**Value**
`x < y`	false
`x > y`	
`x >= y`	
`x == y`	
`x != y`	

Assertions

- the relops are used to make assertions that evaluate to either true or false
  - `5 < 10` is a true assertion
  - `10 < 5` is a false assertion

- an expression formed using a relop is a Boolean expression
- a Boolean expression is either true or false

- let `x = 10;` and `y = 7;`

Expression	**Value**
`x < y`	false
`x > y`	true
`x >= y`	
`x == y`	
`x != y`	

Assertions

- the relops are used to make assertions that evaluate to either true or false
  - `5 < 10` is a true assertion
  - `10 < 5` is a false assertion

- an expression formed using a relop is a Boolean expression
- a Boolean expression is either true or false

- let `x = 10;` and `y = 7;`

Expression	**Value**
`x < y`	false
`x > y`	true
`x >= y`	true
`x == y`	
`x != y`	

Assertions

- the relops are used to make assertions that evaluate to either true or false
  - `5 < 10` is a true assertion
  - `10 < 5` is a false assertion

- an expression formed using a relop is a Boolean expression
- a Boolean expression is either true or false

- let `x = 10;` and `y = 7;`

Expression	**Value**
`x < y`	false
`x > y`	true
`x >= y`	true
`x == y`	false
`x != y`	

Assertions

- the relops are used to make assertions that evaluate to either true or false
  - `5 < 10` is a true assertion
  - `10 < 5` is a false assertion

- an expression formed using a relop is a Boolean expression
- a Boolean expression is either true or false

- let `x = 10;` and `y = 7;`

Expression	**Value**
`x < y`	false
`x > y`	true
`x >= y`	true
`x == y`	false
`x != y`	true

Returned Values

- because a relop returns a value of true or a value of false, we can assign that returned value to a variable
- what kind of variable can store the values true and false?

Returned Values

- because a relop returns a value of true or a value of false, we can assign that returned value to a variable
- what kind of variable can store the values true and false?
- variables of type bool

```
unsigned x = 10;
unsigned y = 7;

bool value1 = x < 5; // value1 is now false
bool value2 = x != y; // value2 is now true
```

Outputting Boolean Values

- by default, cout displays Boolean values in the old-fashioned C way
- true is printed on screen as 1, while false is printed as 0
- this is very poor form

- instead, we use the boolalpha io manipulator (sticky)
- to print the actual Boolean values

see program print_boolean.cpp, based on Checkpoint 4.1 on page 155

Comparing Floating Point Values

```
int main()
{
  double a = 1.5;
  double b = a + 0.00000000000000001;

  bool equal = (a == b);
  cout << boolalpha << equal << endl;
  return 0;
}
```

- what appears on the screen?

Comparing Floating Point Values

```
int main()
{
  double a = 1.5;
  double b = a + 0.00000000000000001;

  bool equal = (a == b);
  cout << boolalpha << equal << endl;
  return 0;
}
```

- what appears on the screen?
- true!

Comparing Floating Point Values

- in fact, the program does not compile correctly:

```
compare_floating.cpp:10:19: warning: comparing
    floating point with == or != is unsafe
```

- the four lt and gt relops are fine with floating point values
- but you cannot test floating point values with $==$ or $!=$

Comparing Floating Point Values

- I weigh a chemical sample and find its weight to be 3.306587 grams
- you weigh a sample and find it weighs 3.306421 grams
- do the two samples weigh the same?

Comparing Floating Point Values

- I weigh a chemical sample and find its weight to be 3.306587 grams
- you weigh a sample and find it weighs 3.306421 grams
- do the two samples weigh the same?

- all measurement on a floating-point scale is approximate
- and, many floating-point values cannot be stored exactly in the computer's floating-point format
- you the programmer must decide how close is "close enough"
- we use the term EPSILON to express "close enough"
- in the weighing problem above, maybe "close enough" is within 0.001 grams

Comparing Floating Point Values

```
int main()
{
  const double EPSILON = 0.001;
  double weight1 = 3.306587;
  double weight2 = 3.306421;

  bool equal = abs(weight1 - weight2) <= EPSILON;
  cout << boolalpha << equal << endl;
  return 0;
}
```

- no warnings
- correct answer
- for each situation, you decide an appropriate EPSILON

Bit Patterns

- every value in a computer is stored as some pattern of bits
- for an unsigned short, the value 25 is stored as the bit pattern 0000 0000 0001 1001
- for a char, the value 'Q' is stored as the bit pattern 0101 0001
- there isn't really a 'Q' in the variable, we just agree that 0101 0001 will stand for 'Q'
- for a bool
  - the value true is stored as the bit pattern 0000 0001
  - the value false is stored as 0000 0000

Truth Values

- going the other direction, regardless of what it represents in its true data type
- every bit pattern can be interpreted as a simple unsigned integer
- the bit pattern 0101 0001 can be interpreted as 81: $2^6 + 2^4 + 2^0$
- so the bit pattern 0101 0001 means two different things: 'Q' as a char, and 81 as an unsigned
- indeed, in the ASCII chart, you will find the decimal value of 'Q' is listed as 81

- in exactly the same way, true is stored internally as 0000 0001, which is decimal 1 when interpreted as an unsigned integer
- and false is stored internally as 0000 0000

Truth Values

BUT!

- false is not the "same thing" as 0, as Gaddis implies
- false is a bool value, while 0 is an integer value
- the two are different types
- even worse, Gaddis says you can use any non-zero value, such as -123, as true — UGH!
- how on Earth does -123 reasonably equal true?
- it doesn't but Gaddis says it does
- (and unfortunately, C++ lets you use it that way) — BAD!
- use true to represent true, and use `boolalpha` to cause true to be printed as "true", not as 1

Program Flow

- all of the programs we have seen so far are sequential
- the statements are executed one after another, from beginning to end
- they start with the first statement after the opening curly brace of main
- and end with the `return 0;` just before the closing curly brace of main
- one statement follows another, without exception

Conditional Execution

- the real purpose of the relops is to allow conditional execution of statements
- instead of deterministic execution in which one statement follows another without exception
- in conditional execution, a statement might be executed
- or it might not be executed

- whether it is executed or not depends on a Boolean expression formed from a relop

The if Statement

- the mechanism that C++ (and most other languages) uses for conditional execution is the

if statement

- the if statement is used to interrupt the sequential flow of program statements
- the if statement is used to conditionally execute a specific set of program statements

- maybe yes, maybe no, depending ...

see program4_2.cpp, from Gaddis page 158

The if Statement Structure

```
if (expression)
{
  statement;
  statement;
  ...
}
```

- if is a reserved word
- expression is a Boolean expression using relops enclosed in parentheses
- the braces delimit the body of the if statement
- multiple statements (one or more) form the body of the if statement; they are indented
- thus the if statement is a compound statement
- there is no semicolon after the closing brace — the closing brace itself ends the if statement

Problems

- Gaddis lists a bunch of things that you can do wrong with an if statement

```
if (a = b);
    foo;
    bar;
```

semicolon

assignment instead of equality

missing braces