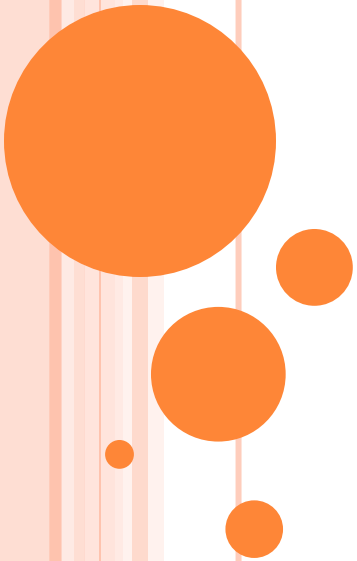

Conditional Statements

CS A150 - C++ Programming 1

The `if` Statement

- Used to implement a decision
- Has two parts: a *condition* and a *body*
- Example:

```
if (area < 0)
        cerr << "Error: Negative area\n";
```

- Body is true ***only*** if condition is true

Comparison Operators

- Used to compare numbers *and* strings

C++	Math Notation	Description
>	>	Greater than
>=	≥	Greater than or equal
<	<	Less than
<=	≤	Less than or equal
==	=	Equal
!=	≠	Not equal

Comparison Operators (cont.)

- Return a ***boolean*** (true or false)
- In C++
  - any **zero** value is ***false***
  - all others are ***true***

Common Error

- Operators
  - `=` is used for assignment
  - `==` tests for equality
- Be careful!

```
if (x = 7) …
```

  - Will *always* be true
  - x will have the value 7 afterwards

THE `if` STATEMENT (CONT.)

- Body may have multiple statements:

```
if (area < 0)
{
        cout << "Error: Negative area\n";
        length = 0;
}
```

- **Must** be enclosed in curly braces { }
  - Called a *block statement*
- Executed sequentially
- Curly braces may be used with a single statement
  - It is *recommended*

The `if/else` Statement

- Checks a condition: If true performs an action, otherwise it performs another action.
  - Add an else followed by a statement, or a block:

```
if (area >= 0)
{
    cout << "The side length is " << sqrt(area) << "\n";
}
else
{
    cerr << "Error: Negative area\n";
}
```

Syntax: `if` Statement

if (*condition*)
    $statement_1$

if (*condition*)
    $statement_1$
else if (condition)
    $statement_2$
...
else
    $statement_3$

Example:

```
if (x == 0)
   cout << "x is equal to 0" << endl;
else if (x == 1)
   cout << "x is equal to 1" << endl;
…
else
   cerr << "x is a negative number" <<
   endl;
```

Purpose: Execute a statement if the condition is true. When paired with **else**, execute the next statement if the condition is false.

Example

Example:

```
if (x >= 0)
   cout << "A";
if (x == 2)
   cout << "B";
else if (x == 3)
   cout << "C";
else
   cout << "D";
```

- What is the output for
  - x = -1 ?

Example

Example:

```
if (x >= 0)
   cout << "A";
if (x == 2)
   cout << "B";
else if (x == 3)
   cout << "C";
else
   cout << "D";
```

- What is the output for
  - x = -1 ? D

Example

Example:

```
if (x >= 0)
   cout << "A";
if (x == 2)
   cout << "B";
else if (x == 3)
   cout << "C";
else
   cout << "D";
```

- What is the output for
  - x = -1 ? D
  - x = 2 ?

Example

Example:

```
if (x >= 0)
   cout << "A";
if (x == 2)
   cout << "B";
else if (x == 3)
   cout << "C";
else
   cout << "D";
```

- What is the output for
  - x = -1 ? D
  - x = 2 ? AB

Example

Example:

```
if (x >= 0)
   cout << "A";
if (x == 2)
   cout << "B";
else if (x == 3)
   cout << "C";
else
   cout << "D";
```

- What is the output for
  - x = -1 ? D
  - x = 2 ? AB
  - x = 0 ?

Example

Example:

```
if (x >= 0)
   cout << "A";
if (x == 2)
   cout << "B";
else if (x == 3)
   cout << "C";
else
   cout << "D";
```

- What is the output for
  - x = -1 ? D
  - x = 2 ? AB
  - x = 0 ? AD

Multiple Alternatives

- Nested branching can be used to
  evaluate complex expressions.
- Once a test succeeds,
  no other tests are performed
- The final else is a default action,
  if no preceding test is true.

Multiple Alternatives (cont.)

- Consider translating a value on the letter-grade scale:

```
if (score >= 90.0)
    cout << 'A';
else if (score >= 80.0)
    cout << 'B';
else if (score >= 70.0)
    cout << 'C';
else if (score >= 60.0)
    cout << 'D';
else
    cout << 'F';
```

Multiple Alternatives (cont.)

- *Order* is important
- This does ***not*** work as intended:

```
if (score >= 60.0)
     cout << 'D';
else if (score >= 70.0)
     cout << 'C';
else if (score >= 80.0)
     cout << 'B'
else if (score >= 90.0)
      cout << 'A';
else
      cout << 'F';
```

Multiple Alternatives (order)

- Independent if statements (without the else) are **inefficient** in this example:

```
if (score == 5)
    cout << 'A';
if (score == 4)
    cout << 'B';
if (score == 3)
    cout << 'C';
if (score == 2)
    cout << 'D';
if (score == 1)
    cout << 'F';
```

The compiler will check each one of them even if the first one is true.

The switch Statement

- Can be used in place of if/else statements that compare a *single integer value* for equality:

```
int digit;
...
switch(digit)
{
case 1:
   digit_name = "one";
   break;
case 2:
   digit_name = "two";
   break;
…
default:
   digit_name = "";
   break;
}
```

The switch Statement (cont.)

- Test cases must be integer or char types
- Cases are tested in order
- Use the break statement to *get out* of the switch
- No faster in modern compilers
- Only use if improves readability

Common Error

- The dangling else problem
- Consider this code:

```
double shippingCharge = 5.00;
if (country == "USA")
        if (state == "HI")
                shippingCharge = 10.00;
 else   // Pitfall!
        shippingCharge = 20.00;
```

- Indentation suggests that the else is paired with the test country == "USA"
- **BUT `else`** is *always* matched to the ***preceding*** **`if`**

Boolean Operations

- Boolean operations have boolean (***true/false***) expressions as operands
- Modify or combine conditions into larger, more complex Boolean expressions
- Two binary operations in C++:
  - **&&**
    - Logical ***and*** is *true* when both operands are *true*
  - **||**
    - Logical **or** is *true* when *either* operand is *true*
  - **!**
    - Logical **not** inverts the value of any expression

Evaluating Boolean Expression

Display 2.2 **Truth Tables**

AND

Exp_1	*Exp_2*	*Exp_1* && *Exp_2*
true	true	true
true	false	false
false	true	false
false	false	false

OR

Exp_1	*Exp_2*	*Exp_1* || *Exp_2*
true	true	true
true	false	true
false	true	true
false	false	false

NOT

Exp	!(*Exp*)
true	false
false	true

Boolean Operations (examples)

- The following test is true only when both country is "USA", and the weight is < 1

```
if (country == "USA" && weight < 1)
        shippingCharge = 2.50;
```

- This test is true if state is either "HI" or "AK":

```
if (state == "HI" || state == "AK")
        shippingCharge = 10.00;
```

Boolean Operators (lazy evaluation)

- These two operators are evaluated *lazily*
- Operands are evaluated left-to-right
- Evaluation ends as soon as the truth value is determined
- Given X is *true* and Y is *false*:

```
X || Y
```

is *true*; Y is never evaluated

```
Y && X
```

is *false*; X is never evaluated

Common Error

- The following will produce an error

```
if ( 1 <= x <= 5 )
        // Error makes sense in mathematics
```

- Needs to be done this way:

```
if ( 1 <= x  &&  x <= 5 )
```

The Selection Operator

- Also called the **conditional** operator
- C++'s ***only*** *ternary* operator:

```
test ? value1 : value2
```

- It *is* an expression (has a value)
- Example:

```
y = (x >= 0) ? 2 : 6;
```

is equivalent to

```
if (x >= 0)
      y = 2;
else
      y = 6;
```

Questions?

(Conditional Statements)