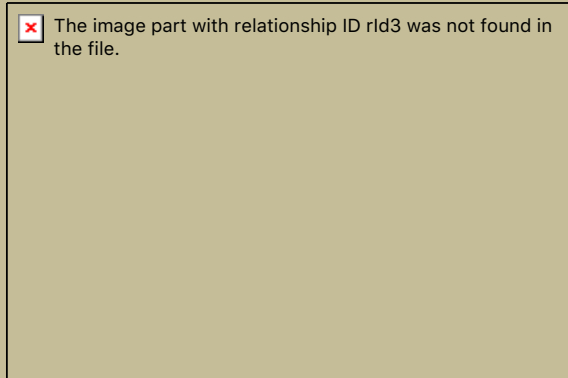


COMPUTER SCIENCE 1: STARTING COMPUTING CSCI 1300



Ioana Fleming / Vipra Gupta
Spring 2018
Lecture 8



University of Colorado
Boulder

Agenda

- Today
 - Conditional Statements – if-else
- Next Week
 - While, Do While loops
 - Strings



Announcements

- Rec 3 due on 2/3
- Hmwk 2 due on 2/4
- Hmwk 3 due 2/11
- Practicum 1 has been scheduled:
 - February 21st, 2018
 - In lecture. 50 minutes. Bring a laptop!
 - Review: in lecture on 2/19
 - Visible in *Tentative Schedule* on Moodle



Control Flow: Learning Objectives

- Boolean Expressions
 - Building, Evaluating & Precedence Rules
- Branching Mechanisms
 - if-else
 - switch
 - Nesting if-else
- Loops
 - While, do-while, for
 - Nesting loops



Boolean Expressions

- Data type bool
 - Returns true or false
 - true, false are predefined library consts



Boolean Expressions: Comparison Operators

1. Comparison Operators: `==`, `<`, `>`, `!=`, `<=`, `>=`

2. Logical Operators

- Logical AND (`&&`)
- Logical OR (`||`)

Display 2.1 Comparison Operators

MATH SYMBOL	ENGLISH	C++ NOTATION	C++ SAMPLE	MATH EQUIVALENT
<code>=</code>	Equal to	<code>==</code>	<code>x + 7 == 2*y</code>	$x + 7 = 2y$
<code>≠</code>	Not equal to	<code>!=</code>	<code>ans != 'n'</code>	$ans \neq 'n'$
<code><</code>	Less than	<code><</code>	<code>count < m + 3</code>	$count < m + 3$
<code>≤</code>	Less than or equal to	<code><=</code>	<code>time <= limit</code>	$time \leq limit$
<code>></code>	Greater than	<code>></code>	<code>time > limit</code>	$time > limit$
<code>≥</code>	Greater than or equal to	<code>>=</code>	<code>age >= 21</code>	$age \geq 21$

Evaluating Boolean Expressions: Truth Tables

Display 2.2 Truth Tables

AND

<i>Exp_1</i>	<i>Exp_2</i>	<i>Exp_1</i> && <i>Exp_2</i>
true	true	true
true	false	false
false	true	false
false	false	false

OR

<i>Exp_1</i>	<i>Exp_2</i>	<i>Exp_1</i> <i>Exp_2</i>
true	true	true
true	false	true
false	true	true
false	false	false

NOT

<i>Exp</i>	!(<i>Exp</i>)
true	false
false	true



Precedence Examples

- Arithmetic before logical
 - $x + 1 > 2 \ || \ x + 1 < -3$ means:
 - $(x + 1) > 2 \ || \ (x + 1) < -3$
- Short-circuit evaluation
 - $(x \geq 0) \ \&\& \ (y > 1)$
 - Be careful with increment operators!
 - $(x > 1) \ \&\& \ (y++)$ *// don't do it!*
- Integers as boolean values
 - All non-zero values \rightarrow true
 - Zero value \rightarrow false



Branching Mechanisms

- if-else statements

- Choice of two alternate statements based on condition expression

- Example:

```
if (temp > 60)
    cout << "It's warm outside" << endl;
else
    cout << "It's cold outside" << endl;
```



if-else Statement Syntax

- Formal syntax:

```
if (<boolean_expression>
    <yes_statement>
else
    <no_statement>
```

- Note: in example above, each alternative is only ONE statement!



Compound/Block Statement

- Only "get" one statement per branch
- Must use compound statement { }
for multiples
 - Also called a "block" statement
- Each block should have block statement
 - Even if just one statement
 - Enhances readability



Compound Statement in Action

- Note indenting in this example:

```
if (myScore > yourScore)
{
    cout << "I win!\n";
    wager = wager + 100;
}
else
{
    cout << "I wish these were golf scores.\n";
    wager = 0;
}
```



Common Pitfalls

- Operator "=" vs. operator "=="
- One means "assignment" (=)
- One means "equality" (==)

```
if (x = 12)    ←Note operator used!  
    Do_Something  
else  
    Do_Something_Else
```

Let's go to C9 and see what happens!



The Optional else

- else clause is optional
 - If, in the false branch (else), you want "nothing" to happen, leave it out
 - Example:

```
if (sales >= minimum)
    salary = salary + bonus;
cout << "Salary = " << salary;
```
 - Note: nothing to do for false condition, so there is no else clause!
 - Execution continues with cout statement



Boolean Return-Type Functions

- Function return-type can be any valid type
 - Given function declaration/prototype:
`bool appropriate(int rate);`
 - And function's definition:

```
bool appropriate (int rate)
{
    return (((rate>=10) && (rate<20)) || (rate==0));
}
```
 - Returns "true" or "false"
 - Function call, from some other function:

```
if (appropriate(entered_rate))
    cout << "Rate is valid\n";
```



Nested Statements

- if-else statements contain smaller statements
 - Compound or simple statements (we've seen)
 - Can also contain any statement at all, including another if-else statement!

- Example:

```
if (speed > 55)
    if (speed > 80)
        cout << "You're really speeding!";
    else
        cout << "You're speeding.";
```

- Note proper indenting!



Multiway if-else

- Not new, just different indenting
- Avoids "excessive" indenting
 - Syntax:

Multiway if-else Statement

SYNTAX

```
if (Boolean_Expression_1)
    Statement_1
else if (Boolean_Expression_2)
    Statement_2
    .
    .
    .
else if (Boolean_Expression_n)
    Statement_n
else
    Statement_For_All_Other_Possibilities
```



Multiway if-else Example

EXAMPLE

```
if ((temperature < -10) && (day == SUNDAY))  
    cout << "Stay home.";  
else if (temperature < -10) //and day != SUNDAY  
    cout << "Stay home, but call work.";  
else if (temperature <= 0) //and temperature >= -10  
    cout << "Dress warm.";  
else //temperature > 0  
    cout << "Work hard and play hard.";
```

The Boolean expressions are checked in order until the first true Boolean expression is encountered, and then the corresponding statement is executed. If none of the Boolean expressions is *true*, then the *Statement_For_All_Other_Possibilities* is executed.



The switch Statement

- A statement for controlling multiple branches
- Can do the same thing with if statements but sometimes switch is more convenient
- Uses controlling expression which returns bool data type (true or false)
- Syntax:
 - Next slide



switch Statement Syntax

switch Statement

SYNTAX

```
switch (Controlling_Expression)
{
    case Constant_1:
        Statement_Sequence_1
        break;
    case Constant_2:
        Statement_Sequence_2
        break;
        .
        .
        .
    case Constant_n:
        Statement_Sequence_n
        break;
    default:
        Default_Statement_Sequence
}
```

*You need not place a **break** statement in each case. If you omit a **break**, that case continues until a **break** (or the end of the **switch** statement) is reached.*

The controlling expression must be integral! This includes char.



The switch Statement in Action

EXAMPLE

```
int vehicleClass;
double toll;
cout << "Enter vehicle class: ";
cin >> vehicleClass;

switch (vehicleClass)
{
    case 1:
        cout << "Passenger car.";
        toll = 0.50;
        break;
    case 2:
        cout << "Bus.";
        toll = 1.50;
        break;
    case 3:
        cout << "Truck.";
        toll = 2.00;
        break;
    default:
        cout << "Unknown vehicle class!";
}
```

*If you forget this **break**,
then passenger cars will
pay \$1.50.*



The switch: multiple case labels

- Execution "falls thru" until break

- switch provides a "point of entry"

- Example:

- case 'A':

- case 'a':

- cout << "Excellent: you got an "A"!\n";
 - break;

- case 'B':

- case 'b':

- cout << "Good: you got a "B"!\n";
 - break;

- Note multiple labels provide same "entry"



switch Pitfalls/Tip

- Forgetting the break;
 - No compiler error
 - Execution simply "falls thru" other cases until break;
- Biggest use: MENUs
 - Provides clearer "big-picture" view
 - Shows menu structure effectively
 - Each branch is one menu choice

!!! No “menu” built-in function exists in C++



switch Menu Example

- Switch statement "perfect" for menus:

```
switch (response)
{
    case 1:
        // Execute menu option 1
        break;
    case 2:
        // Execute menu option 2
        break;
    case 3:
        // Execute menu option 3
        break;
    default:
        cout << "Please enter valid response.";
}
```



Conditional Operator

- Also called "ternary operator"
 - Allows embedded conditional in expression
 - Essentially "shorthand if-else" operator
 - Example:

```
if (n1 > n2)
    max = n1;
else
    max = n2;
```
 - Can be written:

```
max = (n1 > n2) ? n1 : n2;
```

 - "?" and ":" form the "ternary" operator

