Advanced Selection

CS1A

- if-then-else-if
- switch statements
- Complex Boolean expressions
  - Boolean Operators
- Boolean Data type

Selection Structures

What if I only want some instructions to run some of the time?

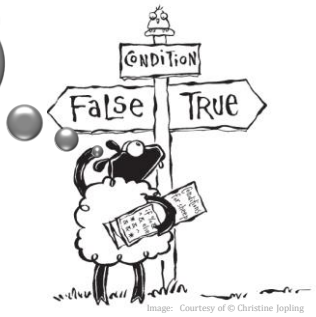


Image: Courtesy of © Christine Jopling

Selection Structures

Selection
➔ Choosing between two or more alternative actions

- Run certain instructions based on some condition
- Conditions are based on Boolean Expressions
  - An expression that evaluates to 1 of 2 possibilities
  - Either True or False
- The computer evaluates a Boolean Expression and determines which instructions to execute based on the result
- Boolean expressions are formed using relational operators

3-types of Selection Statements

- One-way Decisions
  - If-Then Statements
- Two-way Decisions
  - If-Then-Else Statements

  OR
  - *Conditional Statements*
- Multi-way Decisions
  - Nested If-Then or Nested If-Then-Else Statements

  OR
  - *If-Then-Else-If*
  - *Switch statements*

Today we will focus on if-then-else-if, switch statements, & the conditional operator.

If-Then-Else-If

Nested if's are one way of coding multi-way decisions.

If-Then-Else-If statements are also multi-way decisions.

Syntax:

```
if ( logical expression )
{
    stmtT1;
}
else if ( logical expression )
{
    stmtT2;
}
. . .
else if ( logical expression )
{
    stmtTN;
}
else
{
    stmtF;
}
```

Executes if the first logical expression is true → Otherwise it drops to the next else if

This executes if the 1st logical expression was false and the 2nd logical expression was true → Otherwise it drops to the next else if and so on

This executes if all the previous logical expressions evaluated to false

Example – If-Then-Else-If

Lets say you wanted to output what class a user is in based on a variable called classCode.

classCode is of type char and represents the following values:
F = freshman
S = sophomore
J = junior
R = senior

```
if (classcode == 'F')
{
    cout << "You are a freshman!"<< endl;
}
else if (classcode == 'S')
{
    cout << "You are a sophomore";
{
else if (classcode == 'J')
{
    cout <<"You are a junior";
}
else if (classcode == 'R')
{
    cout << "You are a senior";
}
else
{
    cout << "Invalid classcode";
}
```

It is a good practice to handle unexpected inputs

Switch Statement

- Allows for multi-way selection
- Eliminates the need for many nested ifs

Syntax:

```
switch ( expression )
{
        case constant-expression : statement ;
        ...
        default : statement;
}
```

- If the expression evaluates to the constant- expression then the appropriate statement(s) is executed
- Otherwise the default statement is executed

Switch Example

```
switch (classCode)
{
   case 'F' : cout << "You are a freshman";
              break;
   case 'S' : cout << "You are a sophomore";
              break;
   case 'J' : cout << "You are a junior";
   case 'R' : cout << "You are a senior";
              break;
   default : cout << "You entered an invalid code";
              break; // not necessary but a good habit
}
```

What will happen if classCode == 'J'?

Break Statement

- Break statement prevents the case statement from following through.
- It can be useful in some situations

```
switch (classCode)
{
    case 'F' :
    case 'f' : cout << "You are a freshman";
               break;
    case 'S' :
    case 's' : cout << "You are a sophomore";
               break;
    etc...
```

This will execute when classCode == 'F' or 'f'. Statements succeeding a case will execute until the break is reached

Break Statement

- The break statement forces a block of code to exit (or terminate).
- If you don't break in a switch statements all of the statements succeeding a case will execute!
- This can be useful if you want the same code to execute under multiple cases (or situations).

WARNING:
Switch statements are the only time you should use the break statement.
It is considered bad practice to use it in a loop or if statement!!

If-Then-Else-If vs. Switch

Use a switch statement when

- you need to make several comparisons using the SAME variable

Use an IF-Then-Else-IF when…

- You have only a few comparisons or
- You need to check different variables

Unique statements only please

What is wrong with this?

```
if (classcode=='F')
{
   studentCount = studentCount + 1;
   cout << "You are a freshman!"<< endl;
}
else if (classcode=='S')
{
   studentCount = studentCount + 1;
   cout << "You are a sophomore";
{
else if (classcode=='J')
{
    studentCount = studentCount + 1;
   cout <<"You are a junior";
}
else if … etc
```

Conditional Operator

- Shortcut to create a simple if-then-else statement

Syntax:
condition? true_statements: false_statements;

Example:

```
overTime = (hrsWkd > 40 ? (hrsWkd-40)*rate*1.5: 0.0);
```

NOTE: the assignment goes first

Condition to be Tested precedes the?

If true this statement is executed

If false this statement is executed

What would the value of overtime be if:
hrsWkd = 40 and rate = 10 ?
hrsWkd = 60 and rate = 10 ?

Note:
This can also be used with a cout << statement

You can nest them too

Let's say hourly employees get 1½ * their rate for overtime, but other employees just get regular pay.

```
overTime =  (hrsWkd > 40  ? (hourly == 'T' ? (hrsWkd - 40)* rate * 1.5
                                           : (hrsWkd - 40)* rate)
                          : 0.0);
```


Conditional Operator

```
if (price > 3.0)
{
   cout << "over 3";
}
else
{
    if (price == 3.0)
    {
       cout << "equals 3";
    }
    else
    {
       cout << "less than 3";
    }
}
```

Convert this using the Conditional Operator

Final Notes

- Can an if-then-else (or else-if) structure can replace any switch statement?
- Can a switch statement replace any if-then-else-if structure?
- Switch statements are based off the same variable
  - Best used if there are many unique conditions
- Which statement should you use for a two-way statement?
  - If-else or the conditional operator (?:)

NOTE: The statements within any selection structure should be unique to that condition!

Comparing Floating Point Values

- Floating point values are a little trickier to compare than integers
- This is because of the way they are stored in memory
  - They are rounded so it is rare that they will evaluate to be the same (even if you evaluate them to be the same)
- Thus, we just want to check if they are "close enough" to call them equal
- One method
  - First calculate the absolute value of the difference of the two numbers
  - Second, check it against some very small epsilon value

Example

```
const float EPSILON = 0.00001;
float val1, val2;

if (fabs(val1 - val2) < EPSILON)
{
   cout << "values are equal";
}
```

EPSILON is some value which we determine will be within a "close enough" margin
In this case it is 0.00001

fabs
→A C++ library function that returns the absolute value of a floating point expression.

Boolean Operators

CS1A

- Complex Boolean Expressions
  - NOT operator (!)
  - AND operator (&&)
  - OR operator (||)
- Bool datatype

Boolean Operators

- Up until now we have used relational operators to construct Boolean expressions
- What if we want to check two or more conditions in one statement?

EXAMPLE

We want to execute a set of statements if the grade entered is 'a' or 'A'

We want to execute a set of statements if the weight entered is between 100 and 300

We can use Boolean Operators to test many conditions in one expression

Relational Operators

We use Relational Operators to form Boolean Expressions

==	Equal
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal
!=	Not Equal

We use Boolean Operators to compare several sets values at once.

Logical Boolean Operators & Expressions

Boolean Operators are used to form complex decision statements

Function	Syntax	# of operands
NOT	!	Unary
AND	&&	Binary
OR	||	Binary

We use these operators in conjunction with relational operators (<, >, ==, etc.)

George Boole:
Developed Boolean logic and published it in 1847 in his book, "The Mathematical Analysis of Logic"
This led to the field of symbolic logic.

NOT (!)

NOT (denoted as !)

- When it precedes a single expression it gives the opposite result.
- Unary operator (only needs 1 operand)

Example

hrsWkd <= 40

!(hrsWkd > 40)

Truth Table

Value	after !
T(1)	F(0)
F(0)	T(1)

Negating the Relational Operators

!(a == b) ⇔
!(a != b) ⇔
!(a >= b) ⇔
!(a > b) ⇔
!(a <= b) ⇔
!(a < b) ⇔

AND (&&)

AND (Denoted as &&)

- Combines two logical expressions and requires that both expressions be true for the entire expression to be true

Example

num1 <= 10 && num1 != 5

This expression would be TRUE only when both conditions are TRUE

num1 <= 1 AND num1 did not = 5

Operand1 (num1 <= 10)	Operand2 (num1 != 5)	&&
T(1)	T(1)	T(1)
T(1)	F(0)	F(0)
F(0)	T(1)	F(0)
F(0)	F(0)	F(0)

Only time && is TRUE is if they both are true

OR (||)

OR (Denoted as ||)

- combines two logical expressions and states that if either or both of the expressions are TRUE, the entire expression is TRUE

Example

num1 < 1 || num1 > 10

This expression would be FALSE only when both conditions are FALSE

num1 >= 1 AND num1 <= 10

Exp1 (num1 < 1)	Exp22 (num1 > 10)	||
T(1)	T(1)	T(1)
T(1)	F(0)	T(1)
F(0)	T(1)	T(1)
F(0)	F(0)	F(0)

Only time || is FALSE is if they both are false

DeMorgan's law

- Lets say that exp1 & exp2 are boolean expressions so they evaluate to TRUE of FALSE

Note: The Symbols Change

!(exp1 || exp2) ⇔ !exp1 && !exp2

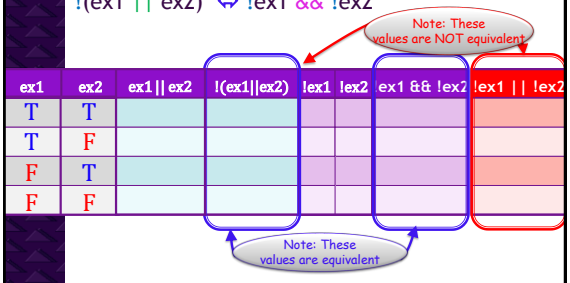
!(exp1 && exp2) ⇔ !exp1 || !exp2

Remember that...

Exp1	Exp2	II	&&
T(1)	T(1)	T(1)	T(1)
T(1)	F(0)	T(1)	F(0)
F(0)	T(1)	T(1)	F(0)
F(0)	F(0)	F(0)	F(0)

DeMorgan's Law #1

!(ex1 || ex2) ⇔ !ex1 && !ex2



ex1	ex2	ex1 || ex2	!(ex1||ex2)	!ex1	!ex2	ex1 && !ex2	!ex1 || !ex2
T	T						
T	F						
F	T						
F	F						

Order of Precedence (Expanded)

()
++ --
!
* / %
+ -
< <= > >=
== !=
&&
||
=

Highest

Lowest

Exercises

Rewrite the expressions distributing the !

- !(num1 == num2)
  ⇔
- !(num1 == num2 || num1 == num3)
  ⇔
- !(num1 == num2 && num3 > num4)
  ⇔
- !(num1 >= num2 && num3 < num4 || num1 != -1)
  ⇔

Exercise

- Write a do while loop that checks classCode for valid inputs 'F','S','J','R'
  - ➔ Distribute any !'s in your boolean expression

Math notation is not C++ syntax

5 < x < 15 is okay in math

This is not equivalent to 5 < x < 15 in C++

How would C++ view this?

Write the equivalent in C++

C++ uses Short-Circuit Evaluation

- **Short Circuit Evaluation** refers to how a language evaluates logical expressions
- Left to Right order
- When using an AND (&&) operator evaluation **stops** as soon as FALSE condition is found
- When using an OR (||) operator evaluation **stops** as soon as a TRUE condition is found

Declaring a Boolean Variable

The Boolean data type can be assigned one of two values: true or false.

Syntax:
bool *variableName*;

EXAMPLE

```
bool dataOK;
int   int1;
cout << "Enter in an integer: ";
cin  >> int1;
dataOK = int1 >= 10;
if (dataOK)
{
   cout << "all is good";
}
else
{
   cout << "value is too low";
} ...
```

How would you check if dataOK was false?

Declaring a Boolean Variable

```
int in1, in2, in3, in4;
bool bl1, bl2;
...
bl1 = in1 > in2;
bl2 = in3 > in4;

if (bl1 && bl2)
{
   cout << "The bl1 & bl2 are true") << endl;
}
else if (bl1 || bl2)
{
   cout << "Either the bl1 is true or the bl2 is true" << endl;
}
cout << bl1<< '\t' << bl2;
```

What would output if →
in1 = 32, in2 = 4, in3 = 45, in4 = 5

in 1 = 32, in2 = 4, in3 = 5, in4 = 45

in1 = 4, in2 = 4, in3 = 5, in4 = 45

in1 = 42, in2 = 4, in3 = 5, in4 = 45

Exercise - revised

- Write a do while loop to validate inputs for the class code.
  Assign the appropriate boolean expression into the boolean variable below

```
bool invalid;
```