Booleans and Selection

Recall that C++ has a `bool` type for storing Boolean values

- `true`
- `false`

Example

```cpp
bool isEmpty = name.empty();
```
Comparisons

- Used to express logical conditions.
- Several standard operators for comparisons
  - `==`, equality
  - `!=`, not equal
  - `<`, `>`
  - `<=`, less than or equal to
  - `>=`, greater than or equal to

Comparisons (Example)

- Problem: Suppose a voter’s name and their age are given as input separated by a tab. Use Booleans to determine if the voter can vote.
Comparisons (Example)

const int VOTING_AGE = 18;
const char DELIMITER = 't';
string name = ""; // name of voter
int age = -1; // age of voter
bool canVote = false; // can the voter vote

// Get the name and age of the voter
getline(cin, name, DELIMITER);
cin >> age;
// The voter can vote if they are at least as old as
// the voting age.
canVote = age >= VOTING_AGE;

Comparisons (Example)

● Problem: Determine if a person read from input is the President.

const string PRESIDENT = "George W. Bush";
string name = ""; // name of person read
bool isPresident = false; // is the person the President

gtline(cin, name);
isPresident = (PRESIDENT == name);
Logical Operations

- Simple Boolean expressions can be combined together with logical operations
  - &&, and
  - ||, or
  - !, not
- Examples
  
  ```
  canVote = (isCitizen) && (age >= VOTING_AGE);
  cantDrive = age < DRIVING_AGE || licenseRevoked;
  canDrive = !cantDrive;
  ```

Truth Tables

- Use truth tables to track the value of Boolean expressions

| x   | y   | x && y | x || y | !x     |
|-----|-----|--------|--------|--------|
| true| true| true   | true   | false  |
| true| false| false  | true   | false  |
| false| true| false  | true   | true   |
| false| false| false  | false  | true   |
**Precedence**

- Operator precedence table is more complex (See page 509)

<table>
<thead>
<tr>
<th>Precedence Level</th>
<th>Operator Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Left to Right</td>
<td>()</td>
</tr>
<tr>
<td>2) Right to Left</td>
<td>Unary +, Unary -, !</td>
</tr>
<tr>
<td>3) Left to Right</td>
<td>* / %</td>
</tr>
<tr>
<td>4) Left to Right</td>
<td>+ −</td>
</tr>
<tr>
<td>5) Left to Right</td>
<td>&lt; &lt;= &gt; &gt;=</td>
</tr>
<tr>
<td>6) Left to Right</td>
<td>== !=</td>
</tr>
<tr>
<td>7) Left to Right</td>
<td>&amp;&amp;</td>
</tr>
<tr>
<td>8) Left to Right</td>
<td></td>
</tr>
</tbody>
</table>

**Complex Boolean Expressions**

- Use *evaluation trees* to evaluate complex Boolean expressions
  - Best to parenthesize Boolean expressions
- Example

```plaintext
complex = (x != y || y <= z && x > z);
```
Comparing Characters

- Characters can be compared
  - Comparison done with ASCII code
    - `'A' < 'Z'`  // true
    - `'Z' < 'a'`  // true
  - See http://www.asciitable.com/

Comparing Strings

- Strings are compared from left to right, using ASCII codes.
  - One of the operands must be a string named constant or variable.

```java
string fruit = "apple";
string transport = "car";

fruit < "Zebra"    // false
transport < "cars" // true
```
Pitfall

- C++ comparisons are not math comparisons
  - `x < y < z`
    - is syntactically allowed by C++, but will not give you the right answer. Why not?
    - `true` is converted to 1 and `false` is converted to 0
  - So, what happens if `x` is 12, `y` is 25 and `z` is 24?
  - Must use `x < y && y < z` instead.

Selection

- Up until now, the only basic control structure we've used is `sequence`
- Recall the `selection` control structure
Selection in C++

- In C++, selection is written using the *if* statement.
- Syntax:

```c
if (Expression)
    Statement1
else
    Statement2
```

Remember, the inner box means this part is optional.

Selection in C++ (Example)

- Problem: Print out a special message if a student receives an A on their program.

```c
const int LOWEST_A = 90; // lowest grade that gets an A
int progGrade = -1; // init to -1 for debugging

cin >> progGrade;
if (progGrade >= LOWEST_A)
    cout << "Great job!" << endl; // make sure to indent!
```
Selection in C++ (Example)

- Modified Problem: Do the same thing as the previous slide, but print also print out a message if the student doesn't get an A.

```cpp
const int LOWEST_A = 90;
int progGrade = -1;
cin >> progGrade;
if (progGrade >= LOWEST_A)
    cout << "Great Job!" << endl;
else
    cout << "You’ll do better next time!" << endl;
```

Compound Statements

- Sometimes you want to execute more than one statement in an if statement.
- Use braces to make a compound statement.
- Syntax

```cpp
{
    Statement
    Statement
    ...  
}
```
Compound Statement (Example)

- Problem: If a program grade is an A, print out a special message and add one to the total number of A grades, otherwise add one to the total of non-A grades.

```cpp
const int LOWEST_A = 90; cin >> progGrade;
int progGrade = -1; if (progGrade >= LOWEST_A)
    int Agrades = 0;
    otherGrades = 0;
    cout << "Great job!" << endl;
    Agrades = Agrades + 1;
} else
    otherGrades = otherGrades + 1;
```

There's no semicolon

Exercise

```cpp
int age = -1; int attendees = 0;
bool pSup; int sentAway = 0;

// read in values for age and pSup
if (!(age >= MIN_AGE || pSup))
    cout << "You may not enter!" << endl;
    sentAway = sentAway + 1;
} else
    cout << "You may enter." << endl;
    attendees = attendees + 1;
```
Nested If Statements

- Statements
  - Input
  - Output
  - Assignment
  - Compound
  - If
- Because if statements are statements, we can nest them

Nested If Statements (Example)

```cpp
const int MIN_AGE = 17; // Age to enter movie
bool pSup = false; // Parent supervision
int age = -1;

// Read in age and parent supervision information
// Check to see if the viewer can enter the movie
if (age < MIN_AGE)
  if (pSup)
    cout << "You may enter." << endl;
  else
    cout << "You may not enter!" << endl;
else
  cout << "You may enter." << endl;
```
Pitfall (Assignment in Condition)

- C++ converts expressions to boolean values in conditions
  - Nonzero is `true`, zero is `false`
- A common pitfall is to perform an assignment instead of an equality test
  - The value of an assignment statement is the value assigned to the variable

Pitfall (Assignment Example)

- What happens if value read for `change` is 0?

```cpp
int change; // change to return to the customer

cin >> change;
if (change == 0)
{
    cout << "Perfect payment!" << endl;
}
```

- Nothing ever gets printed!
Pitfall (Nested Ifs)

- An `else` clause always combines with the closest `if`.

```
// Failing. will be printed if average is better
// than marginal!
if (average >= PASSING)
    if (average < MARGINAL)
        cout << "Passing but marginal." << endl;
    else
        cout << "Failing." << endl;
```

Pitfall (Nested Ifs)

- Corrected versions

```
if (average >= PASSING)
    if (average < MARGINAL)
        cout << "Passing..." << endl;
    else
        cout << "Failing." << endl;
else
    cout << "Passing..." << endl;
```

Using braces is always a good idea!
Style (Nested Ifs)

- Often if statements are nested in the following way:

```cpp
cout << "You got an ";
if (average >= A_GRADE)
    cout << "A" << endl;
else
    if (average >= B_GRADE)
        cout << "B" << endl;
    else ...
```

Style (Nested Ifs)

- Instead of indenting each `else` clause, the code is formatted as follows:

```cpp
cout << "You got an ";
if (average >= A_GRADE)
    cout << "A" << endl;
else if (average >= B_GRADE)
    cout << "B" << endl;
else if (average >= C_GRADE)
    ...
```
Style (Nested Ifs)

- It's often better to rewrite nested ifs as complex Boolean expressions

```cpp
if (age >= VOTING_AGE)
    if (isCitizen)
        cout << "You may vote." << endl;
```

- Is better written as

```cpp
if (age >= VOTING_AGE && isCitizen)
    cout << "You may vote." << endl;
```

Consider what you would do to print a message if a person couldn't vote.

Switch statements

- A `switch` statement allows you to choose statements to execute based on many possibilities.

- **Expression** must be
  - Integer (`short`, `int`, `long`)
  - Character (`char`)
  - Boolean (`bool`)

```cpp
switch (Expression)
{
    case label1:
        statement1;
        statement1_2
        ...
        break;
    case label2:
        statement2_1
        statement2_2
        ...
        break;
    ...
    default:
        statementd_1
        statementd_2
        ...
        break;
}
```
Switch statements

- **Semantics**
  - If the value of *Expression* is the same as *labeln* then execute `statementn1`, `statementn2`, ..., until a `break` or bottom of `switch` is reached.
  - If no values match, execute statements following `default::`, if the label exists.

```
switch (Expression)
{
    case label1:
        statement1
        statement1
        ...
        break;
    case label2:
        statement2
        statement2
        ...
        break;
    ...
    default:
        statementd
        statementd
        ...
        break;
}
```

### Switch Statement (Example)

- **Problem:** Calculate QCA multiplier based on grade in class (without +/-).
- In this example, if grade is invalid, nothing happens.

```
char grade;   // grade in class
int points = 0;   // QCA multiplier

cin >> grade;
switch (grade)
{
    case 'A':
        points = 4;
        break;
    case 'B':
        points = 3;
        break;
    case 'C':
        points = 2;
        break;
    case 'D':
        points = 1;
        break;
    case 'F':
        points = 0;
        break;
    default:
        break;
}
```
Switch Statement (Example)

- The `break;` statements are optional. In this case the label is used as a starting place.

```cpp
char grade; // grade in class
int points = 0; // QCA multiplier

char grade; // grade in class
int points = 0; // QCA multiplier

char grade; // grade in class
int points = 0; // QCA multiplier

```

Switch Statement (Example)

- The `default:` label is often used to detect errors.

```cpp
switch (grade)
{
    case 'A':
        points = points + 1;
        break;
    case 'B':
        points = points + 1;
    case 'C':
        points = points + 1;
    case 'D':
        points = points + 1;
    case 'F':
        points = points + 1;
    default:
        cout << "Invalid grade: " << grade << endl;
        break;
}
```

```cpp
switch (grade)
{
    case 'A':
        points = points + 1;
        break;
    case 'B':
        points = points + 1;
    case 'C':
        points = points + 1;
    case 'D':
        points = points + 1;
    case 'F':
        points = points + 1;
    default:
        cout << "Invalid grade: " << grade << endl;
        break;
}
```
Switch Statement (Example)

- You can have multiple labels do the same thing.
- Change the previous switch statement to match lower case grades too.

```cpp
char grade; // grade in class
int points = 0; // QCA multiplier

cin >> grade;
switch (grade)
{
    case 'A': case 'a':
        points = points + 1;
        break;
    case 'B': case 'b':
        points = points + 1;
        break;
    case 'C': case 'c':
        points = points + 1;
        break;
    case 'D': case 'd':
        points = points + 1;
        break;
    case 'F': case 'f':
        break;
    default:
        cout << "Invalid grade: " << grade << endl;
}
```

Pitfall

- The value of the expression must match exactly with the label.
  - Switch statements do not support ranges.
  - Have to list every possibility.
- What is printed if temperature is 20? 70?

```cpp
int temperature = 0;

cin >> temperature;
switch (temperature)
{
    case 0:
        cout << "Below freezing." << endl;
        break;
    case 32:
        cout << "Freezing." << endl;
        break;
    case 60: case 85:
        cout << "Getting warm." << endl;
        break;
    default:
        cout << "Hot!" << endl;
}
```