Learning Objectives

• Boolean Expressions
  – Building, Evaluating & Precedence Rules

• Branching Mechanisms
  – if-else
  – switch
  – Nesting if-else

• Loops
  – While, do-while, for
  – Nesting loops

• Introduction to File Input
Boolean Expressions:

Display 2.1 Comparison Operators

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

<table>
<thead>
<tr>
<th>MATH SYMBOL</th>
<th>ENGLISH</th>
<th>C++ NOTATION</th>
<th>C++ SAMPLE</th>
<th>MATH EQUIVALENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to</td>
<td>==</td>
<td>x + y == 2 * y</td>
<td>x + y = y</td>
</tr>
<tr>
<td>≠</td>
<td>Not equal</td>
<td>!==</td>
<td>ans != ‘n’</td>
<td>ans ≠ ‘n’</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>&lt;</td>
<td>count &lt; m + 3</td>
<td>count &lt; m + 3</td>
</tr>
<tr>
<td>≤</td>
<td>Less than or equal to</td>
<td>&lt;=</td>
<td>time &lt;= limit</td>
<td>time ≤ limit</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>&gt;</td>
<td>time &gt; limit</td>
<td>time &gt; limit</td>
</tr>
<tr>
<td>≥</td>
<td>Greater than or equal to</td>
<td>&gt;=</td>
<td>age &gt;= 21</td>
<td>age ≥ 21</td>
</tr>
</tbody>
</table>

Evaluating Boolean Expressions

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

- Truth tables
  - Display 2.2 next slide
### Display 2.2
**Truth Tables**

#### AND

<table>
<thead>
<tr>
<th>Exp₁</th>
<th>Exp₂</th>
<th>Exp₁ &amp;&amp; Exp₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

#### OR

| Exp₁ | Exp₂ | Exp₁ || Exp₂ |
|------|------|--------|
| true | true | true   |
| true | false| true   |
| false| true | true   |
| false| false| false  |

#### NOT

<table>
<thead>
<tr>
<th>Exp</th>
<th>!(Exp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

### Display 2.3
**Precedence of Operators (1 of 4)**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>::</td>
<td>Scope resolution operator</td>
</tr>
<tr>
<td>.</td>
<td>Dot operator</td>
</tr>
<tr>
<td>⇒</td>
<td>Member selection</td>
</tr>
<tr>
<td>[]</td>
<td>Array indexing</td>
</tr>
<tr>
<td>()</td>
<td>Function call</td>
</tr>
<tr>
<td>++</td>
<td>Postfix increment operator (placed after the variable)</td>
</tr>
<tr>
<td>--</td>
<td>Postfix decrement operator (placed after the variable)</td>
</tr>
<tr>
<td>++</td>
<td>Prefix increment operator (placed before the variable)</td>
</tr>
<tr>
<td>--</td>
<td>Prefix decrement operator (placed before the variable)</td>
</tr>
<tr>
<td>!</td>
<td>Not</td>
</tr>
<tr>
<td>-</td>
<td>Unary minus</td>
</tr>
<tr>
<td>+</td>
<td>Unary plus</td>
</tr>
<tr>
<td>*</td>
<td>Dereference</td>
</tr>
<tr>
<td>&amp;</td>
<td>Address of</td>
</tr>
<tr>
<td>new</td>
<td>Create (allocate memory)</td>
</tr>
<tr>
<td>delete</td>
<td>Destroy (deallocate)</td>
</tr>
<tr>
<td>delete[]</td>
<td>Destroy array (dealocate)</td>
</tr>
<tr>
<td>sizeof</td>
<td>Size of object</td>
</tr>
<tr>
<td>(</td>
<td>Type cast</td>
</tr>
</tbody>
</table>

Highest precedence (done first)
### Display 2.3
Precedence of Operators (2 of 4)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Multiply</td>
</tr>
<tr>
<td>/</td>
<td>Divide</td>
</tr>
<tr>
<td>%</td>
<td>Remainder (modulo)</td>
</tr>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>Insertion operator (console output)</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>Extraction operator (console input)</td>
</tr>
</tbody>
</table>

Lower precedence (done later)

---

### Display 2.3
Precedence of Operators (3 of 4)

Display 2.3  
Precedence of Operators

All operators in part 2 are of lower precedence than those in part 1.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>Equal</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>And</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Precedence Examples

- Arithmetic before logical
  - \( x + 1 > 2 \ || \ x + 1 < -3 \) means:
    - \((x + 1) > 2 \ || \ (x + 1) < -3\)

- Short-circuit evaluation
  - \((x >= 0) \ && \ (y > 1)\)
  - Be careful with increment operators!
    - \((x > 1) \ && \ (y++)\)

- Integers as boolean values
  - All non-zero values \(\rightarrow\) true
  - Zero value \(\rightarrow\) false
Strong Enum

• C++11 introduces **strong enums** or **enum classes**
  – Does not act like an integer
  – Examples
    ```cpp
    enum class Days { Sun, Mon, Tue, Wed, Thu, Fri, Sat }
    enum class Weather { Rain, Sun }
    Days d = Days::Tue;
    Weather w = Weather::Sun;
    
    – Illegal: if (d == 0)
    – Legal: if (d == Days::Wed)
    ```

Branching Mechanisms

• if-else statements
  – Choice of two alternate statements based on condition expression
  – Example:
    ```cpp
    if (hrs > 40)
        grossPay = rate*40 + 1.5*rate*(hrs-40);
    else
        grossPay = rate*hrs;
    ```
if-else Statement Syntax

• Formal syntax:
  
  if (<boolean_expression>)
  
  <yes_statement>

  else
  
  <no_statement>

• Note each alternative is only ONE statement!

• To have multiple statements execute in either branch → use compound statement

Compound/Block Statement

• Only "get" one statement per branch

• Must use compound statement { } for multiples
  
  – Also called a "block" stmt

• 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" (==)
  – VERY different in C++!
  – Example:
    if (x = 12) ←Note operator used!
      Do_Something
    else
      Do_Something_Else
The Optional else

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

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 stmt!
  - Example:
    ```cpp
    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) // day != SUNDAY
    cout << "Stay home, but call work."
  else if (temperature <= 0) // and temperature >= -10
    cout << "Dress warm."
  else if (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 (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
}
```

The controlling expression must be integral! This includes char.
The switch Statement in Action

```cpp
EXAMPLE
int vehicleClass;
double toll;
cout << "Enter vehicle class: ";
 cin >> vehicleClass;
switch (vehicleClass)
{
  case 'A':
    case 'a':
      cout << "Excellent: you got an "A"!
      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!";
}
```

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

switch Menu Example

• Switch stmt "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 this "ternary" operator

Loops

• 3 Types of loops in C++
  – while
    • Most flexible
    • No "restrictions"
  – do-while
    • Least flexible
    • Always executes loop body at least once
  – for
    • Natural "counting" loop
while Loops Syntax

Syntax for while and do-while Statements

A while STATEMENT WITH A SINGLE STATEMENT BODY

while (Boolean_Expression)

Statement

A while STATEMENT WITH A MULTISTATEMENT BODY

while (Boolean_Expression)

{
    Statement_1
    Statement_2
    ...
    Statement_Last
}

while Loop Example

• Consider:
  count = 0;               // Initialization
  while (count < 3)        // Loop Condition
  {
      cout << "Hi ";         // Loop Body
      count++;               // Update expression
  }

  – Loop body executes how many times?
do-while Loop Syntax

A do-while STATEMENT WITH A SINGLE-STATEMENT BODY
do
    Statement
while (Boolean_Expression);

A do-while STATEMENT WITH A MULTISTATEMENT BODY
do
    { Statement_1
        Statement_2
        .
        .
        Statement_Last
    } while (Boolean_Expression);

Do not forget the final semicolon.

do-while Loop Example

• count = 0;       // Initialization
do
do {
    cout << "Hi ";     // Loop Body
    count++;          // Update expression
} while (count < 3);       // Loop Condition

– Loop body executes how many times?
– do-while loops always execute body at least once!
while vs. do-while

• Very similar, but...
  – One important difference
    • Issue is "WHEN" boolean expression is checked
    • while: checks BEFORE body is executed
    • do-while: checked AFTER body is executed

• After this difference, they’re essentially identical!

• while is more common, due to it’s ultimate "flexibility"

Comma Operator

• Evaluate list of expressions, returning value of the last expression

• Most often used in a for-loop

• Example:
  first = (first = 2, second = first + 1);
  – first gets assigned the value 3
  – second gets assigned the value 3

• No guarantee what order expressions will be evaluated.
for Loop Syntax

for (Init_Action; Bool_Exp; Update_Action)
Body_Statement

• Like if-else, Body_Statement can be a block statement
  – Much more typical

for Loop Example

• for (count=0;count<3;count++)
  {
    cout << "Hi ";  // Loop Body
  }

• How many times does loop body execute?
• Initialization, loop condition and update all "built into" the for-loop structure!
• A natural "counting" loop
Loop Issues

• Loop’s condition expression can be ANY boolean expression

• Examples:
  
  while (count<3 && done!=0)
  {
    // Do something
  }

  for (index=0;index<10 && entry!=-99)
  {
    // Do something
  }

Loop Pitfalls: Misplaced ;

• Watch the misplaced ; (semicolon)
  – Example:
    while (response != 0) ;
    {
      cout << "Enter val: ";
      cin >> response;
    }
  – Notice the ";" after the while condition!

• Result here: INFINITE LOOP!
Loop Pitfalls: Infinite Loops

• Loop condition must evaluate to false at some iteration through loop
  – If not → infinite loop.
  – Example:
    
    ```cpp
    while (1)
    {
      cout << "Hello ";
    }
    ```
  – A perfectly legal C++ loop → always infinite!

• Infinite loops can be desirable
  – e.g., "Embedded Systems"

The break and continue Statements

• Flow of Control
  – Recall how loops provide "graceful" and clear flow of control in and out
  – In RARE instances, can alter natural flow
• break;
  – Forces loop to exit immediately.
• continue;
  – Skips rest of loop body
• These statements violate natural flow
  – Only used when absolutely necessary!
Nested Loops

- Recall: ANY valid C++ statements can be inside body of loop
- This includes additional loop statements!
  - Called "nested loops"
- Requires careful indenting:
  for (outer=0; outer<5; outer++)
    for (inner=7; inner>2; inner--)
      cout << outer << inner;
  - Notice no { } since each body is one statement
  - Good style dictates we use { } anyway

Introduction to File Input

- We can use cin to read from a file in a manner very similar to reading from the keyboard
- Only an introduction is given here, more details are in chapter 12
  - Just enough so you can read from text files and process larger amounts of data that would be too much work to type in
## Opening a Text File

- Add at the top
  ```cpp
  #include <fstream>
  using namespace std;
  ```
- You can then declare an input stream just as you would declare any other variable.
  ```cpp
  ifstream inputStream;
  ```
- Next you must connect the `inputStream` variable to a text file on the disk.
  ```cpp
  inputStream.open("filename.txt");
  ```
- The “filename.txt” is the pathname to a text file or a file in the current directory.

## Reading from a Text File

- Use
  ```cpp
  inputStream >> var;
  ```
- The result is the same as using `cin >> var` except the input is coming from the text file and not the keyboard.
- When done with the file close it with
  ```cpp
  inputStream.close();
  ```
File Input Example (1 of 2)

- Consider a text file named player.txt with the following text

```
Display 2.10   Sample Text File, player.txt, to Store a Player's High Score and Name

100510
Gordon Freeman
```

File Input Example (2 of 2)

```
Display 2.11   Program to Read the Text File in Display 2.10

1 #include <iostream>
2 #include <fstream>
3 #include <string>
4 using namespace std;
5 int main() {
6    string firstName, lastName;
7    int score;
8    ifstream inputStream;
9    inputStream.open("player.txt");
10   inputStream >> score;
11   inputStream >> firstName >> lastName;
12   cout << "Name: " << firstName << " " << lastName << endl;
13   cout << "Score: " << score << " and " << inputStream.close();
14   return 0;
15 }
```

Sample Dialogue

Name: Gordon Freeman
Score: 100510
Summary 1

- Boolean expressions
  - Similar to arithmetic → results in true or false
- C++ branching statements
  - if-else, switch
  - switch statement great for menus
- C++ loop statements
  - while
  - do-while
  - for

Summary 2

- do-while loops
  - Always execute their loop body at least once
- for-loop
  - A natural "counting" loop
- Loops can be exited early
  - break statement
  - continue statement
  - Usage restricted for style purposes
- Reading from a text file is similar to reading from cin