**Selection Control Structures**

**Outline**
- Boolean Expressions
- Boolean Operators
- Relational Operators
- *if – else* statement

**Control Structures**
- Flow of programming execution control is *sequential* unless a “control structure” is used to change that
- there are 2 general types of control structures:
  - *Selection* (also called branching)
  - *Repetition* (also called looping)
### C++ control structures

- **Selection**
  - if
  - if...else
  - switch
- **Repetition**
  - for loop
  - while loop
  - do...while loop

### Conditional and Logical Expressions

- Control Structures use *conditional* (relational) and logical (Boolean) expressions
- Conditional expressions use *relational operators*: `< <= > >= == !=
- Boolean expressions use *logical operators*: `! && ||`

### bool Data Type

- Result value from either a relational expression or Boolean expression is either `true` or `false`
- In C++, the data type `bool` is a built-in type consisting of just 2 Boolean values, the constants `true` and `false`
- we can declare (Boolean) variables of type `bool` to hold Boolean values `true` or `false`
- In C++, the value `0` represents `false` and any nonzero (`1`) represents `true`
Logical (Boolean) Expressions

A Boolean expression is an expression consists of
- Conditional expressions with relational operators
- a Boolean variable or constant
- Boolean expressions with Boolean operators.

Conditional expressions with relational operators

• Conditional expression is in the form

<table>
<thead>
<tr>
<th>Expression1</th>
<th>Operator</th>
<th>Expression2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score1 &gt; Score2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SideA + SideB == SideC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Length != 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoefB<em>CoefB - 4</em>CoefA*CoefC &gt; 0.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Count &lt;= 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Response == 'Y'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Relational Operators

<table>
<thead>
<tr>
<th>Relational</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than</td>
</tr>
<tr>
<td>==</td>
<td>equal to</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
</tr>
</tbody>
</table>
Example

```cpp
int x, y;
x = 4;
y = 6;
```

<table>
<thead>
<tr>
<th>EXPRESSION</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>x &lt; y</td>
<td>true</td>
</tr>
<tr>
<td>x + 2 &lt; y</td>
<td>false</td>
</tr>
<tr>
<td>x = y</td>
<td>true</td>
</tr>
<tr>
<td>x + 2 &gt;= y</td>
<td>true</td>
</tr>
<tr>
<td>y == x</td>
<td>false</td>
</tr>
<tr>
<td>y == x+2</td>
<td>true</td>
</tr>
<tr>
<td>y == x</td>
<td>4 (true)</td>
</tr>
</tbody>
</table>

Boolean variable or constant

```cpp
bool Done, Flag, Result, Test;
double Score1, Score2, Score3;
int SideA, SideB, SideC;
Done = true;
Flag = false;
Test = Score1 > Score2;
Result = SideA + SideB == SideC;
Result = Done && Flag;
```

Boolean Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>NOT</td>
</tr>
</tbody>
</table>
### AND (&&) Operator

<table>
<thead>
<tr>
<th>Value of X</th>
<th>Value of Y</th>
<th>Value of X &amp;&amp; Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

### OR (||) Operator

| Value of X | Value of Y | Value of X || Y |
|------------|------------|--------------|
| true       | true       | true         |
| true       | false      | true         |
| false      | true       | true         |
| false      | false      | false        |

### Not (!) Operator

<table>
<thead>
<tr>
<th>Value of X</th>
<th>Value of !X</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>
Examples

(\text{Response}=='Y') \text{ || } (\text{Response}=='y')
(\text{Count} > 10) \text{ && } (\text{Response} == 'Y')
!\text{Done}

Example

\begin{tabular}{|c|c|}
\hline
\text{EXPRESSION} & \text{VALUE} \\
\hline
\text{isSenior} & \text{false} \\
\text{hasFever} & \text{true} \\
\text{isSenior \&\& hasFever} & \text{false} \\
\text{isSenior || hasFever} & \text{true} \\
!\text{isSenior} & \text{true} \\
!\text{hasFever} & \text{false} \\
\hline
\end{tabular}

Example

\begin{tabular}{|c|c|}
\hline
\text{EXPRESSION} & \text{VALUE} \\
\hline
!(\text{height} > 60) & \text{true} \\
!(\text{age} < 10) & \text{true} \\
!(\text{height} > 60) & \text{false} \\
\hline
\end{tabular}
Example

```c
int age, height;
age = 25;
height = 70;
```

<table>
<thead>
<tr>
<th>EXPRESSION</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(age &gt; 50) &amp;&amp; (height &gt; 60)</td>
<td>False</td>
</tr>
<tr>
<td>height &gt; 60</td>
<td></td>
</tr>
<tr>
<td>!(height &gt; 60)</td>
<td></td>
</tr>
</tbody>
</table>

Example

```c
int age, weight;
age = 25;
weight = 145;
```

<table>
<thead>
<tr>
<th>EXPRESSION</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>weight &gt; 180 &amp;&amp; (age &gt;= 50)</td>
<td>True</td>
</tr>
</tbody>
</table>

Precedence of Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>Highest precedence</td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>\</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td></td>
</tr>
<tr>
<td>&lt;</td>
<td></td>
</tr>
<tr>
<td>&lt;=</td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td></td>
</tr>
<tr>
<td>&gt;=</td>
<td></td>
</tr>
<tr>
<td>==</td>
<td></td>
</tr>
<tr>
<td>!=</td>
<td></td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operator</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Lowest precedence</td>
</tr>
</tbody>
</table>
Comparing Strings

- two objects of type string (or a string object and a C string) can be compared using the relational operators
- a character by character comparison is made using the ASCII character set values
- if all the characters are equal, then the 2 strings are equal. Otherwise, the string with the character with smaller ASCII value is the “lesser” string

Example

<table>
<thead>
<tr>
<th>string</th>
<th>myState;</th>
<th>string</th>
<th>yourState;</th>
</tr>
</thead>
<tbody>
<tr>
<td>myState</td>
<td>&quot;Texas&quot;;</td>
<td>yourState</td>
<td>&quot;Maryland&quot;;</td>
</tr>
<tr>
<td>myState &lt;= &quot;Texas&quot;</td>
<td>false</td>
<td></td>
<td></td>
</tr>
<tr>
<td>myState &gt; yourState</td>
<td>true</td>
<td></td>
<td></td>
</tr>
<tr>
<td>myState == &quot;Texas&quot;</td>
<td>true</td>
<td></td>
<td></td>
</tr>
<tr>
<td>myState &lt; &quot;Texas&quot;</td>
<td>true</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Write an expression for each

- taxRate is over 25% and income is less than $20000
  \[(\text{taxRate} > 0.25) \land (\text{income} < 20000)\]
- temperature is less than or equal to 75 or humidity is less than 70%
  \[(\text{temperature} <= 75) \lor (\text{humidity} < .70)\]
- age is over 21 and age is less than 60
  \[(\text{age} > 21) \land (\text{age} < 60)\]
- age is 21 or 22
  \[(\text{age} == 21) \lor (\text{age} == 22)\]
**WARNING about Expressions in C++**

- “Boolean expression” means an expression whose value is true or false
- an expression is any valid combination of operators and operands
- each expression has a value
- this can lead to UNEXPECTED RESULTS
- construct your expressions CAREFULLY
- use of parentheses is encouraged
- otherwise, use precedence chart to determine order

**Selection statements**

are used to choose an action depending on the current situation in your program as it is running

**Control Structure**

- A selection statement is a control structure used to (alter the sequential flow of control) choose an action depending on the current situation in your program as it is running.
- if-else statement
**Syntax of if-else Statement**

```cpp
if (Boolean expression)
    Statement1;
else
    Statement2;
```

The else part can be dropped if there is no second choice.

---

**Example**

```cpp
cout<<"Enter your weight";
int Weight;
if (Weight < 300)
    cout<<"You are okay.";
else
    cout<<"You are over Weight.";
```

---

**Use of blocks recommended**

```cpp
if ( Expression )
{
    "if clause"
}
else
{
    "else clause"
}
```
Selection Statement if-else

true

expression

false

if clause else clause

Selection Statement if-else

true

Weight < 300

false

cout<<"You are okay." cout="You are over Weight."

Example

```c
int carDoors, driverAge;
float premium, monthlyPayment;
.
.
if ((carDoors == 4) && (driverAge > 24))
    premium = 650.00;
    cout << " LOW RISK ";
else
    premium = 1200.00;
    cout << " HIGH RISK ";
monthlyPayment = premium / 12.0 + 5.00;
.
.
```

What happens if you omit braces?

```cpp
if ((carDoors == 4) && (driverAge > 24))
    premium = 650.00;
    cout << " LOW RISK " ;
else
    premium = 1200.00 ;
    cout << " HIGH RISK " ;
monthlyPayment = premium / 12.0 + 5.00 ;
```

• COMPILE ERROR OCCURS. The “if clause” is the single statement following the if.

Single statement in if and else clause

Braces can only be omitted when each clause is a single statement

```cpp
if (lastInitial <= 'K')
    volume = 1;
else
    volume = 2;
cout << "Look it up in volume # "
    << volume << " of NYC phone book";
```

Example

• Assign value .25 to discountRate and assign value 10.00 to shipCost if purchase is over 100.00
• Otherwise, assign value .15 to discountRate and assign value 5.00 to shipCost
• Either way, calculate totalBill
**Code for Example**

```cpp
if (purchase > 100.00) {
    discountRate = 0.25;
    shipCost = 10.00;
} else {
    discountRate = 0.15;
    shipCost = 5.00;
}
totalBill = purchase *(1.0 - discountRate) + shipCost;
```

**if Syntax**

```cpp
if (Boolean expression)
    Statement1;
```

• NOTE: Statement can be a single statement, a null statement, or a block.

**Selection Statement if**

![Diagram of if statement flowchart]

true

expression

statement

false
Example

```cpp
cout << "Enter a number";
cin >> Number;
if (Number < 0)
{
    cout << "The number that you enter is negative."
    cout << "Enter another number."
    cin >> Number;
}
```

Terminating your program

```cpp
int number;
cout << "Enter a non-zero number";
cin >> number;
if (number == 0)
{
    cout << "Bad input. Program terminated."
    return 1;
}
else
    // otherwise continue processing
```

Write if or if-else

- If `taxCode` is 'T', increase price by adding `taxRate` times `price` to it.
  ```cpp
  if (taxCode == 'T')
  { 
      price = price + taxRate * price;
  }
  ```
- If `code` has value 1, read values for `income` and `taxRate` from `myInfile`, and calculate and display `taxDue` as their product.
  ```cpp
  if (code == 1)
  {
      myInfile >> income >> taxRate;
      taxDue = income * taxRate;
      cout << taxDue;
  }
  ```
- If `A` is strictly between 0 and 5, set `B` equal to `1/A`, otherwise set `B` equal to `A`.
  ```cpp
  if ((A > 0) && (A < 5))
  { 
      B = 1/A;
  }
  ```
What output? and Why?

```cpp
int age;
age = 20;
if (age = 16)
    { cout << "Did you get driver's license?"; }
```

What output? and Why?

```cpp
int age;
age = 30;
if (age < 18)
    { cout << "Do you drive?";
      cout << "Too young to vote"; }
```

What output? and Why?

```cpp
int code;
code = 0;
if (!code)
    cout << "Yesterday";
else
    cout << "Tomorrow";
```
What output? and Why?

```c
int number;
number = 2;
if (number = 0)
    cout << "Zero value";
else
    cout << "Non zero value";
```

Note

Both the if clause and the else clause of an if-else statement can contain any kind of statement, including another selection statement.

Nested Selection

Multi alternative is also called multi-way branching, and can be accomplished by using NESTED if or if-else statements.
Nested if-else Statement

```cpp
if (Boolean expression)
    Statement1;
else if (Boolean expression)
    Statement2;
else
    Statement3;
```

Nested if Statements

- Each Expression is evaluated in sequence, until some Expression is found that is true.
- Only the specific Statement following that particular true Expression is executed.
- If no Expression is true, the Statement following the final else is executed.
- Actually, the final else and final Statement are optional. If omitted, and no Expression is true, then no Statement is executed.

Nested Selections

```cpp
if (creditsEarned >= 90)
    cout << "SENIOR STATUS ";
else if (creditsEarned >= 60)
    cout << "JUNIOR STATUS ";
else if (creditsEarned >= 30)
    cout << "SOPHOMORE STATUS ";
else
    cout << "FRESHMAN STATUS ";
```
Example

```cpp
if (Average >= 90)
    cout << "Your grade is an A";
else if (Average >= 80)
    cout << "Your grade is a B";
else if (Average >= 70)
    cout << "Your grade is a C";
else if (Average >= 60)
    cout << "Your grade is a D";
else
    cout << "Your grade is an F";
```

Example

```cpp
if (Average >= 90)
    cout << "Your grade is an A";
else if (Average >= 80)
    cout << "Your grade is a B";
else if (Average >= 70)
    cout << "Your grade is a C";
else if (Average >= 60)
    cout << "Your grade is a D";
else
    cout << "Your grade is an F";
```

Writing Nested if Statements

- Display one word to describe the int value of number as “Positive”, “Negative”, or “Zero”

```cpp
if (number > 0)
    cout << "Positive";
else if (number < 0)
    cout << "Negative";
else
    cout << "Zero";
```
Writing Nested if Statements

Your city classifies a pollution index
- less than 35 as "Pleasant",
- 35 through 60 as "Unpleasant",
- and above 60 as "Health Hazard."
Display the correct description of the
pollution index value.

```cpp
if (index < 35)
    cout << "Pleasant";
else if (index <= 60)
    cout << "Unpleasant";
else
    cout << "Health Hazard";
```

Each I/O stream has a state (condition)

- An input stream enters fail state when you
  - try to read invalid input data
  - try to open a file which does not exist
  - try to read beyond the end of the file
- An output stream enters fail state when you
  - try to create a file with an invalid name
  - try to create a file on a write-protected disk
  - try to create a file on a full disk

How can you tell the state?

- The stream identifier can be used as if it were a Boolean variable. It has value false (meaning the last I/O operation on that stream failed) when the stream is in fail state.
- When you use a file stream, you should check on its state.
Checking on the I/O State

```cpp
fstream myOutfile;
myOutfile.open ("A:\myOut.dat",ios::out);
if (! myOutfile) {
    cout << "File opening error.  
          Program terminated."
         << endl;
    return 1;
} // otherwise send output to myOutfile
```

Testing Selection Control Structures

- to test a program with branches, use enough data sets so that every branch is executed at least once
- this is called minimum complete coverage

Testing Often Combines Two Approaches

<table>
<thead>
<tr>
<th>WHITE BOX TESTING</th>
<th>BLACK BOX TESTING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Coverage</td>
<td>Data Coverage</td>
</tr>
<tr>
<td>Allows us to see the program code while designing the tests, so that data values at the boundaries, and possibly middle values, can be tested.</td>
<td>Tries to test as many allowable data values as possible without regard to program code.</td>
</tr>
</tbody>
</table>
How to Test a Program

- design and implement a test plan
- a test plan is a document that specifies the test cases to try, the reason for each, and the expected output
- implement the test plan by verifying that the program outputs the predicted results

<table>
<thead>
<tr>
<th>PHASE</th>
<th>RESULT</th>
<th>TESTING TECHNIQUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem solving</td>
<td>Algorithm</td>
<td>Algorithm walkthrough</td>
</tr>
<tr>
<td>Implementation</td>
<td>Coded program</td>
<td>Code walkthrough, Trace</td>
</tr>
<tr>
<td>Compilation</td>
<td>Object program</td>
<td>Compiler messages</td>
</tr>
<tr>
<td>Execution</td>
<td>Output</td>
<td>Implement test plan</td>
</tr>
</tbody>
</table>