Goals

• Recognize when to use the Guarded Action pattern
• Implement the Guarded Action pattern with the if statement
• use relational operators such as < and >
• create and evaluate expressions with the logical operators
• use bool objects
• understand the Alternative Action pattern
• implement the Alternative Action pattern with the C++ if...else statement
• implement the Multiple Selection n with if...else and switch
• solve problems using the Multiple Selection pattern
Why do we need selection?

• Programs must often anticipate a variety of situations
• Consider an Automated Teller Machine:
  • ATMs must serve valid bank customers.
  • They must also reject invalid PINs
  • The code that controls an ATM must permit different requests
  • Software developers must implement code that anticipates all possible transactions
Selective Control

- Programs often contain statements that may not always execute
- Sometimes a statement may execute and other certain conditions it may not
  - Reject invalid PIN entries at an ATM instead of allowing a withdrawal
- We say an action is guarded from executing
## The Guarded Action Pattern

<table>
<thead>
<tr>
<th>Pattern:</th>
<th>Guarded Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem:</td>
<td>Execute an action only under certain conditions</td>
</tr>
</tbody>
</table>
| General Form: | if( logical-expression )  
|              | true-part              |
| Code Example: | if(aStudent.GPA() >= 3.5)  
|              | deanList.push_back(aStudent); |
The if statement

- The if is the first statement that alters strict sequential control. General form
  
  
  ```
  if ( logical-expression )
  true-part ;
  ```

- `logical-expression`: any expression that evaluates to nonzero (true) or zero (false)
- In C++, almost everything is true or false
• After the logical expression of the if statement evaluates, the true-part executes only if the logical expression is true.
Example if statement

double hours = 38.0;
// Add 1.5 hours for hours>40.0 (overtime)
if (hours > 40.0)
    hours = 40.0 + 1.5 * (hours - 40.0);

• What is the value of hours when hours is
  double hours = 38.0;  // _______
double hours = 40.0;  // _______
double hours = 42.0;  // _______
Another way

• The if statement could also be written with a block

```java
double hours = 42.0;
if (hours > 40.0) {
    hours = 40.0 + 1.5 * (hours - 40.0);
}
```

• Sometimes the block is required  consider using `{ }

```java
if (hours > 40.0) {
    regularHours = 40.0;
    overtimeHours = hours - 40.0;
}
```
Relational Operators

- Logical expressions often use relational operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal</td>
</tr>
<tr>
<td>==</td>
<td>Equal</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal</td>
</tr>
</tbody>
</table>
Logical Expressions

• Which expressions are true, which are false?

```java
int n1 = 78;
int n2 = 80;

n1 < n2     // _____
n1 >= n2     // _____
(n1 + 35) > n2 // _____
n1 > 78      // _____
n1 == n2     // _____
n1 != n2     // _____
```
Logical Expressions with strings

- Which expressions are true, which are false?

```csharp
string s1 = "Carson";
string s2 = "Carly";

s1 < s2
s1 > s2
s1 == s2
s1 != s2
s1 > s2
s2 < "C"
```
Relational Operators in \textit{if} Statements

\begin{verbatim}
double x = 59.0;
if (x >= 60.0) {
    cout << "passing";
}
if (x < 60.0) {
    cout << "failing";
}
\end{verbatim}

- What is the output when \texttt{x} is 59, 60, and 61?

\begin{verbatim}
double x = 59.0;  \_\_\_\_\_
double x = 60.0;  \_\_\_\_
double x = 61.0;  \_\_\_\_
\end{verbatim}
• Using = for == is a common mistake. For example the following two statements are legal, but ...

```cpp
int x = 25;
// Because assignment statements evaluate
// to the expression on the right of =, x=1
// is always 1, which is nonzero, or true
if (x = 1) // should be (x == 1)
    cout << "I'm always displayed";
```

• So consider putting the literal first

```cpp
if (1 = x) // This is a compiletime error
```
The Alternative Action Pattern

- Programs often contain statements that select between one set of actions or another
- Examples
  - withdraw or deposit money
  - pass or fail the entrance requirements
- This is the Alternative Action Pattern
  - choose between two alternate sets of actions
## Alternative Action

<table>
<thead>
<tr>
<th>Pattern:</th>
<th>Alternative Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem:</td>
<td>Must choose one action from two alternatives</td>
</tr>
<tr>
<td>Outline:</td>
<td><code>if (true-or-false-condition is true) action-1</code> else <code>action-2</code></td>
</tr>
<tr>
<td>Code</td>
<td><code>if(finalGrade &gt;= 60.0) cout &lt;&lt; &quot;passing&quot; &lt;&lt; endl; else cout &lt;&lt; &quot;failing&quot; &lt;&lt; endl;</code></td>
</tr>
</tbody>
</table>
### if-else

```plaintext
if (logical-expression)
  true-part;
else
  false-part;
```

- When the logical expression evaluates to true, the true-part executes and the false-part is disregarded.
- When the logical expression is false, only the false-part executes.
The if...else statement

- The if...else statement allows two alternate courses of action

```
logical expression
```

```
statement-1
```

```
statement-n
```

```
statement-1
```

```
statement-n
```
if...else Example

```cpp
if (miles > 24000)
    cout << "Tune-up " << miles-24000 << " miles overdue";
else
    cout << "Tune-up due in " << 24000-miles << " miles";
```

<table>
<thead>
<tr>
<th>miles</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>30123</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>24000</td>
<td></td>
</tr>
</tbody>
</table>
The Block {} with if-else

• Blocks may be used even when

```cpp
if (miles > 24000) {
    cout << "Tune-up " << miles-24000 << " miles overdue";
} else {
    cout << "Tune-up due in " << 24000-miles << " miles";
}
```

• Using curly braces all the time helps avoid difficult to detect errors
bool Objects

• The standard bool type stores one of two values true and false

• A bool object stores the result of a logical expression:

```cpp
bool ready = false;
cout << ready << endl; // 0 for false
double hours = 4.5;
ready = hours >= 4.0;
cout << ready << endl; // 1 for true
```
It is common to have functions that return one of the bool values (true or false)

```cpp
// true if n is odd
bool odd(int n) {
    return (n % 2) != 0;
}

// Use the odd function
int main() {
    int anInt = 3;
    if( odd(anInt) )
        anInt++;
    cout << anInt;  // 4
    return 0;
}
```
Boolean Operators

• A logical operator (&& means and) used in an if...else statement

```cpp
int test = 50;
if( (test >= 0) && (test <= 100) )
    cout << "Test is in range";
else
    cout << "**Warning--Test out of range";
```

• The code describes whether or not the value of `test` is in the range of 0 through 100 inclusive.
**Truth Tables for Boolean Operators**

- Truth tables for the Logical (Boolean) operators
  
  ![ (not)   || (or)   && (and)  

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
<th>Expression</th>
<th>Result</th>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>! false</td>
<td>true</td>
<td>true</td>
<td></td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>! true</td>
<td>false</td>
<td>true</td>
<td></td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td></td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false &amp; &amp; true</td>
</tr>
<tr>
<td>false</td>
<td></td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false &amp; &amp; false</td>
</tr>
</tbody>
</table>

- You can also use these more readable operators instead of  ![ !   ||   &&  

  not    or    and
More Precedence Rules

• The following slide summarizes all operators used in this textbook (we've seen them all now)
• Precedence: most operators are evaluated (grouped) in a left-to-right order:
  \[ \frac{a}{b}/c/d \] is equivalent to \[ (((a/b)/c)/d) \]
• Assignment operators group in a right-to-left order so the expression
  \[ x=y=z=0 \] is equivalent to \[ x=(y=(z=0)) \]
<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Grouping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:: ()</td>
<td>Scope resolution, Function call</td>
<td>Left to right</td>
</tr>
<tr>
<td>Unary</td>
<td>!, +, -</td>
<td>Right to left</td>
</tr>
<tr>
<td>Multiplicative</td>
<td>* / %</td>
<td>Left to right</td>
</tr>
<tr>
<td>Additive</td>
<td>+, -</td>
<td>Left to right</td>
</tr>
<tr>
<td>Input/Output</td>
<td>&gt;&gt; &lt;&lt;</td>
<td>Left to right</td>
</tr>
<tr>
<td>Relational</td>
<td>&lt;, &gt;, &lt;=, &gt;=</td>
<td>Left to right</td>
</tr>
<tr>
<td>Equality</td>
<td>==, !=</td>
<td>Left to right</td>
</tr>
<tr>
<td>and</td>
<td>&amp;&amp;</td>
<td>Left to right</td>
</tr>
<tr>
<td>or</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assignment</td>
<td>=</td>
<td>Right to left</td>
</tr>
</tbody>
</table>
Applying Operators and Precedence Rules

• Use the precedence rules to evaluate the following expression:

```c
int j = 5;
int k = 10;
bool TorF;

TorF = (j * (1 + k) > 55) ||
      ((j + 5 <= k) && (j > k));
```

• What is assigned to `TorF`? _______
#include "Grid.h" // for class Grid

// Return true if the mover is at an end of the world
bool moverOnEdge(const Grid & aGrid) {
    return(    aGrid.row()==0     // on north edge?
              || aGrid.row()==aGrid.nRows()-1 // on south?
              || aGrid.column()==0     // on west edge?
              || aGrid.column()==aGrid.nColumns()-1 );
}

int main() {
    Grid tarpit(5, 10, 4, 4, east);
    if( moverOnEdge(tarpit) )
        cout << "On edge" << endl; // On edge
    else
        cout << "Inside border" << endl;
    return 0;
}
Short Circuit Boolean Evaluation

• C++ logical expressions evaluate sub-expressions in a left to right order
• Sometimes the evaluation can stop early
• This will never evaluates $\sqrt{x}$ of a negative number:
  \[
  \text{if}((x \geq 0.0) \land (\sqrt{x} \leq 2.5))
  \]
• $test > 100$ will not be evaluated when $test$ is negative
  \[
  \text{if}(test < 0 \lor test > 100)
  \]
A bool member function

- Consider changing \texttt{BankAccount::withdraw} so it only withdraws money if the balance is sufficient
- Also have it return \texttt{true} in this case
- Have it return \texttt{false} when there are insufficient funds, after the change to the state of the object
- First change heading in class \texttt{BankAccount} that is in the file \texttt{BankAccount.h}

```cpp
bool withdraw(double withdrawalAmount);
```
a bool member function

• Also change implementation in BankAccount.cpp

```cpp
bool BankAccount::withdraw(double amount) {
    // post: return true if withdrawal was successful or false with insufficient funds
    if (balance >= amount) {
        balance = balance - amount;
        return true;
    }
    return false;
}
```
Multiple Selection

• *Nested logic:* When one control structure contains another similar control structure
  • an if else inside another if else
  • allows selections from 3 or more alternatives
• We must often select one alternative from many
<table>
<thead>
<tr>
<th><strong>Pattern:</strong></th>
<th>Multiple Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Problem:</strong></td>
<td>Must execute one set of actions from three or more alternatives.</td>
</tr>
</tbody>
</table>
| **Outline:** | if \((condition\ 1\ is\ true)\)  
execute action 1  
else if \((condition\ 2\ is\ true)\)  
execute action 2  
// ... 
else if \((condition\ n-1\ is\ true)\)  
execute action \(n-1\)  
else  
execute action \(n\) |
| **Code Example:** | if(grade < 60)  
result = "F";  
else if(grade < 70)  
result = "D";  
else if(grade < 80)  
result = "C";  
else if(grade < 90)  
result = "B";  
else  
result = "A"; |
Example of Multiple Selection nested if...else

if (GPA < 3.5) {
    cout << "Try harder" << endl;
} else {
    if (GPA < 4.0) {
        cout << "Dean's List";
    } else {
        cout << "President's list";
    }
    The false part is another if...else
}

GPA          Output:
3.0          
3.6          
4.0          

Multiple Returns

• It's possible to have multiple return statements in a function terminate when the first return executes

```cpp
string letterGrade(double percentage) {
    if (percentage >= 90)
        return "A";
    if (percentage >= 80)
        return "B";
    if (percentage >= 70)
        return "C";
    if (percentage >= 60)
        return "D";
    return "F";  // percentage < 0
}
```
Testing Multiple Selection

• It is often difficult and unnecessary to test every possible value *imagine all those doubles 0.1, 0.001, 0.0001,...*

• Testing our code in "most" branches can prove dangerously inadequate

• Each branch through the multiple selection should be tested
Perform Branch Coverage Test

- To correctly perform branch coverage testing we need to do the following:
  - Establish a set of data that ensures all paths will execute the statements after the logical expressions
  - Execute the code call the function with the nested logic for all selected data values
  - Observe that the code behaves correctly for all data compare program output with expected results
  - This is glass box testing when you look at the code
Boundary Testing

• Boundary testing involves executing the code using the boundary (cutoff) values
• What grade would you receive with a percentage of 90 using this code

```csharp
string letterGrade(double percentage) {
    if (percentage > 90)
        return "A";
    if (percentage >= 80)
        return "B";
    ...
```
function assert

• So far testing has been done by printing with cout
• This requires a careful inspection of the cout statements and the associated output
• C++ has an assert function takes a bool argument to more easily test our functions
  • If the argument is false, C++ will inform you with a line of output that begins with Assertion failed
  • In this case, assert will terminate the program
  • If all expressions in all calls to the assert function are true, there is no output
• Consider this test driver that uses assert
  • If letterGrade is correct there will be no output

```c
int main() {
    assert("A" == letterGrade(100.0));
    assert("A" == letterGrade(90.0));
    assert("B" == letterGrade(89.9));
    assert("B" == letterGrade(80.0));
    assert("C" == letterGrade(79.9));
    assert("C" == letterGrade(70.0));
    assert("D" == letterGrade(69.9));
    assert("D" == letterGrade(60.0));
    assert("F" == letterGrade(59.0));
    assert("F" == letterGrade(59.9));
}
```
function assert

• If any assertion is wrong, you will get a message
  • The program terminates, the 3rd assert is not executed

```c
int main() {
    assert("A" == letterGrade(100.0));
    assert("E" == letterGrade(90.0));
    assert("A" == letterGrade(59.9));
}
```

Assertion failed: ("E" == letterGrade(90.0)),
function main, file ../src/testGrade.cpp, line 29.
The switch Statement

switch (switch-expression) {
    case value-1:
        statement(s)-1
        break;       // many cases are allowed
    case value-n:
        statement(s)-n
        break;
    default:
        default-statement(s)
}
Switch control

• When a switch statement is encountered:
  • the switch-expression is evaluated. This value is compared to each case value until switch-expression equals the case value.
  • All statements after the colon : are executed.
• It is important to include the break statement
• The switch expression must evaluate to one of C++'s integral types
  int  char  enum
char Objects

- A char object stores 1 character
  
  'A'  'x'  'c'  '?'  ' '  '1'  '.'

- Or 1 escape sequence

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>'\n'</td>
<td>new line</td>
</tr>
<tr>
<td>'&quot;'</td>
<td>double quote in a char</td>
</tr>
<tr>
<td>'\'</td>
<td>single quote in a char</td>
</tr>
<tr>
<td>'/'</td>
<td>forward slash</td>
</tr>
<tr>
<td>'\t'</td>
<td>tab</td>
</tr>
</tbody>
</table>
char option = '?';
cout << "Enter W)ithdraw  D)eposit B)alances: ";
cin >> option;
switch (option) {
case 'W':
    cout << "Withdraw" << endl;
    break;
case 'D':
    cout << "Deposit" << endl;
    break;
case 'B':
    cout << "Balance" << endl;
    break;
default:
    cout << "Invalid" << endl;
} // end switch

Show output when

option == '?' _______
option == 'W' _______
option == 'B' _______
option == 'A' _______
option == 'Q' _______