Selection

- Relational and Logical Operators and Expressions
- Control Structures
- if and if-else statements
- Nested if
Where we are Headed – decision making

int minimum(int x, int y) {
    int min;
    if (x > y) {
        min = y;
    } else {
        min = x;
    }
    return min;
}
A Note about \textit{true} and \textit{false}

- use \texttt{bool} type for variables with values
  - \texttt{true} or \texttt{false}
A Note about *true* and *false*

- use **bool** type for variables with values
  - *true* or *false*

- in C++, when expressions are evaluated,
  - zero value is considered false,
  - any non-zero value is considered true
Relational & Logical Operators

- Relational Operators
  
  `<`  `<=`  `>`  `>=`  `==`  `!=`

- Logical Operators
  
  `&&` (and)  `||` (or)  `!` (not)
Example Relational Expressions

```cpp
bool undergrad = true;
int age = 20;
char gender = 'F';

<table>
<thead>
<tr>
<th>EXPR</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>undergrad</td>
<td>true</td>
</tr>
<tr>
<td>age &lt;= 25</td>
<td>true</td>
</tr>
<tr>
<td>gender == 'M'</td>
<td>false</td>
</tr>
<tr>
<td>gender = 'M'</td>
<td>true ('M')</td>
</tr>
<tr>
<td>(age % 2) == 0</td>
<td>true</td>
</tr>
</tbody>
</table>
```

(Also: gender ← ‘M’)

---

EECS 183
University of Michigan

Lecture 05
**AND Operation (&&)**

- Truth table:

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$A$ &amp;&amp; $B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
**OR Operation (||)**

- Truth table:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
### NOT Operation (!)

- **Truth table:**

<table>
<thead>
<tr>
<th></th>
<th>!A</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>
## Precedence Rules Recap

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>ASSOCIATIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HIGH</strong></td>
<td></td>
</tr>
<tr>
<td>( )</td>
<td>left to right</td>
</tr>
<tr>
<td>! + -</td>
<td>unary; right to left</td>
</tr>
<tr>
<td>* / %</td>
<td>left to right</td>
</tr>
<tr>
<td>+ -</td>
<td>left to right</td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>left to right</td>
</tr>
<tr>
<td>== !=</td>
<td>left to right</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>right to left</td>
</tr>
</tbody>
</table>

**LOW**
Example Logical Expressions

bool undergrad = true;
int age = 20;
char gender = 'F';

<table>
<thead>
<tr>
<th>EXPR</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>!undergrad</td>
<td></td>
</tr>
<tr>
<td>undergrad &amp;&amp; (age &gt; 18)</td>
<td></td>
</tr>
<tr>
<td>undergrad &amp;&amp; (age == 18)</td>
<td></td>
</tr>
<tr>
<td>(17 &lt; age) &amp;&amp; (age &lt; 24)</td>
<td></td>
</tr>
<tr>
<td>(compare to &quot;17 &lt; age &lt; 24&quot;)</td>
<td></td>
</tr>
<tr>
<td>undergrad</td>
<td></td>
</tr>
<tr>
<td>(undergrad</td>
<td></td>
</tr>
</tbody>
</table>
Quiz

• Determine the value, true or false, for each

```c
int count = 0, limit = 10;
```

a) \((\text{count} == 0) \&\& (\text{limit} < 20)\)

b) \((\text{count} == 0) \&\& \text{limit} < 20\)

c) \((\text{limit} > 20) \mid\mid (\text{count} < 5)\)

d) \(! (\text{count} == 12)\)

e) \((\text{count} == 1) \&\& (x < y)\)

f) \((\text{count} < 10) \mid\mid (x < y)\)

g) \(! ((\text{count} < 10) \mid\mid (x < y)) \&\& (\text{count} >= 0)\)

h) \(((\text{limit} / \text{count}) > 7) \mid\mid (\text{limit} < 20)\)

i) \((\text{limit} < 20) \mid\mid ((\text{limit}/\text{count}) > 7)\)

j) \((5 \&\& 7) + (!6)\)
Quiz

• Determine the value, true or false, for each

```c
int count = 0, limit = 10;
```

a) `(count == 0) && (limit < 20)`
b) `count == 0 && limit < 20`
c) `(limit > 20) || (count < 5)`
d) `! (count == 12)`
e) `(count == 1) && (x < y)`
f) `(count < 10) || (x < y)`
g) `!( (count < 10) || (x < y) ) && (count >= 0)`
h) `((limit / count) > 7) || (limit < 20)`
i) `(limit < 20) || ((limit/count) > 7)`
j) `(5 && 7) + (!6)`
Quiz

• Determine the value, true or false, for each

```
int count = 0, limit = 10;

a)  (count == 0) && (limit < 20)
b)  count == 0 && limit < 20
c)  (limit > 20) || (count < 5)
d)  ! (count == 12)
e)  (count == 1) && (x < y)
f)  (count < 10) || (x < y)
g)  !((count < 10) || (x < y)) && (count >= 0)
h)  ((limit / count) > 7) || (limit < 20)
i)  (limit < 20) || ((limit/count) > 7)
j)  (5 && 7) + (!6)
```
Quiz

- Determine the value, true or false, for each

```c
int count = 0, limit = 10;
```

a) `(count == 0) && (limit < 20)`
b) `count == 0 && limit < 20`
c) `(limit > 20) || (count < 5)`
d) `!(count == 12)`
e) `(count == 1) && (x < y)`
f) `(count < 10) || (x < y)`
g) `!(count < 10) || (x < y) && (count >= 0)`
h) `(limit / count) > 7 || (limit < 20)`
i) `(limit < 20) || ((limit/count) > 7)`
j) `(5 && 7) + (!6)`
Quiz

• Determine the value, true or false, for each

```cpp
int count = 0, limit = 10;

a)  (count == 0) && (limit < 20)
b)  count == 0 && limit < 20
c) (limit > 20) || (count < 5)
d) ! (count == 12)
e) (count == 1) && (x < y)
f) (count < 10) || (x < y)
g) !((count < 10) || (x < y)) && (count >= 0)
h) ((limit / count) > 7) || (limit < 20)
i) (limit < 20) || ((limit/count) > 7)
j) (5 && 7) + (!6)
```
Quiz

• Determine the value, true or false, for each

```java
int count = 0, limit = 10;
```

a) (count == 0) && (limit < 20)
b) count == 0 && limit < 20
c) (limit > 20) || (count < 5)
d) ! (count == 12)
e) (count == 1) && (x < y)
f) (count < 10) || (x < y)
g) !( (count < 10) || (x < y) ) && (count >= 0)
h) ((limit / count) > 7) || (limit < 20)
i) (limit < 20) || ((limit/count) > 7)
j) (5 && 7) + (!6)
Quiz

• Determine the value, true or false, for each

```cpp
int count = 0, limit = 10;
```

a) `(count == 0) && (limit < 20)`  
b) `count == 0 && limit < 20`  
c) `(limit > 20) || (count < 5)`  
d) `! (count == 12)`  
e) `(count == 1) && (x < y)`  

f) `(count < 10) || (x < y)`  
g) `!( (count < 10) || (x < y) ) && (count >= 0)`  
h) `((limit / count) > 7) || (limit < 20)`  
i) `(limit < 20) || ((limit/count) > 7)`  
j) `(5 && 7) + (!6)`
Quiz

• Determine the value, true or false, for each

```c
int count = 0, limit = 10;
a)  (count == 0) && (limit < 20)
b)  count == 0 && limit < 20
c)  (limit > 20) || (count < 5)
d)  ! (count == 12)
e)  (count == 1) && (x < y)
f)  (count < 10) || (x < y)
g)  !((count < 10) || (x < y)) && (count >= 0)
h)  ((limit / count) > 7) || (limit < 20)
i)  (limit < 20) || ((limit/count) > 7)
j)  (5 && 7) + (!6)
```
Quiz

• Determine the value, true or false, for each

```c
int count = 0, limit = 10;
```

a) \((\text{count} == 0) \&\& (\text{limit} < 20)\)
b) \(\text{count} == 0 \&\& \text{limit} < 20\)
c) \((\text{limit} > 20) \mid\mid (\text{count} < 5)\)
d) \(! (\text{count} == 12)\)
e) \((\text{count} == 1) \&\& (\text{x} < \text{y})\)
f) \((\text{count} < 10) \mid\mid (\text{x} < \text{y})\)
g) \(! ((\text{count} < 10) \mid\mid (\text{x} < \text{y})) \&\& (\text{count} >= 0)\)
h) \((\text{limit} / \text{count}) > 7) \mid\mid (\text{limit} < 20)\)
i) \((\text{limit} < 20) \mid\mid ((\text{limit} / \text{count}) > 7)\)
j) \((5 \&\& 7) + (!6)\)
Quiz

• Determine the value, true or false, for each

```c
int count = 0, limit = 10;
```

a) \((\text{count} == 0) \land (\text{limit} < 20)\)
b) \((\text{count} == 0) \land (\text{limit} < 20)\)
c) \((\text{limit} > 20) \lor (\text{count} < 5)\)
d) \!(\text{count} == 12)\)
e) \((\text{count} == 1) \land (x < y)\)
f) \((\text{count} < 10) \lor (x < y)\)
g) \!(((\text{count} < 10) \lor (x < y)) \land (\text{count} >= 0)\)
h) \(((\text{limit} \div \text{count}) > 7) \lor (\text{limit} < 20)\)
i) \((\text{limit} < 20) \lor ((\text{limit}/\text{count}) > 7)\)
j) \((5 \land 7) + (\!6)\)
Quiz

- Determine the value, true or false, for each

```c
int count = 0, limit = 10;
```

a) `(count == 0) && (limit < 20)`
b) `count == 0 && limit < 20`
c) `(limit > 20) || (count < 5)`
d) `!(count == 12)`
e) `(count == 1) && (x < y)`
f) `(count < 10) || (x < y)`
g) `!(count < 10) || (x < y) && (count >= 0)`
h) `((limit / count) > 7) || (limit < 20)`
i) `(limit < 20) || ((limit/count) > 7)`
j) `(5 && 7) + (!6)`
Basic Control Structures

- A **sequence** is a series of statements that execute one after another
- A **selection(branch)** statement is used to determine which of two different statements to execute depending on certain conditions
- A **looping(repetition)** statement is used to repeat statements while certain conditions are met
- A **subprogram** is a smaller part of another program; a collection of subprograms solves the original problem
SEQUENCE

Statement → Statement → Statement → ...
SELECTON(branch)

IF Condition THEN Statement1 ELSE Statement2
LOOP(repetition)

WHILE Condition DO Statement1

Condition

Statement(s)

False

True

...
Function_1

a meaningful collection of SEQUENCE, SELECTION, LOOP, function
if statement ("if-then")

```plaintext
if ( <cond-expression> )
  <statement>
```

- statement executes if and only if (iff) conditional expression is true
const float DISCOUNT = 0.30;
float price;
char usedCode;

cout << "Enter list price of book: ";
cin >> price;
cout << "Is it used? Y or N: ";
cin >> usedCode;

if ( usedCode == 'Y' ) // calculate sale price
{
    price = price - (DISCOUNT * price);
}
cout << "Selling price $" << price << endl;
Example Program

const float DISCOUNT = 0.30;
float price;
char usedCode;

cout << "Enter list price of book: ";
cin >> price;
cout << "Is it used? Y or N: ";
cin >> usedCode;

if ( usedCode == 'Y' ) // calculate sale price
    price = price - (DISCOUNT * price);

cout << "Selling price $" << price << endl;
const float DISCOUNT = 0.30;
float price;
char usedCode;

cout << "Enter list price of book: ";
 cin >> price;
cout << "Is it used? Y or N: ";
cin >> usedCode;

if ( usedCode == 'Y' ) // calculate sale price
{
    price = price - (DISCOUNT * price);
}
cout << "Selling price $" << price << endl;

 Example Program
if ( usedCode == 'Y' )
{
    cout << "Applying used discount" << endl;
    price = price - (DISCOUNT * price);
}
cout << "Selling price $" << price << endl;
if ( usedCode == 'Y' )
    cout << "Applying used discount" << endl;
    price = price - (DISCOUNT * price);

    cout << "Selling price $" << price << endl;
if ( usedCode == 'Y' )
    cout << "Applying used discount" << endl;
price = price - (DISCOUNT * price);

cout << "Selling price $" << price << endl;
if ( usedCode == 'Y' )
{
    cout << "Applying used discount" << endl;
    price = price - (DISCOUNT * price);
}
cout << "Selling price $" << price << endl;
Example Case-Insensitive

```cpp
if (( usedCode == 'Y' ) || ( usedCode == 'y' ))
{
    cout << "Applying used discount" << endl;
    price = price - (DISCOUNT * price);
}
cout << "Selling price $" << price << endl;
```
Example Case-Insensitive

```cpp
if (( usedCode == 'Y' ) || ( usedCode == 'y' ))
{
    cout << "Applying used discount" << endl;
    price = price - (DISCOUNT * price);
}
cout << "Selling price $" << price << endl;
```

what if we changed || to && ?
Example Case-Insensitive

```c++
if (( usedCode == 'Y' ) || ( usedCode == 'y' ))
{
    cout << "Applying used discount" << endl;
    price = price - (DISCOUNT * price);
}
cout << "Selling price $" << price << endl;
```

what if we changed || to && ?

expression always false
if-else statement
"if-then-else"

if (<cond-expr>)
  <statement1>
else
  <statement2>
if ( myScore > yourScore )
{
    cout << "I win!\n" << endl;
    wager = wager + 100;
}
else // handle the other case
{
    cout << "I wish these were golf scores.\n";
    wager = wager / 2;
}
cout << "wager is $" << wager << endl;
• Note the placement of the brackets

```latex
{ 
  ...
}
```
What prints?

```c
x = 13;

if ( x = 12 )
{
    cout << "Here" << endl;
}
else
{
    cout << "There" << endl;
}
```
**Nesting**

- Statement in then or else clause may itself be an if
  ```plaintext
  if ( <cond-expr> )
    <statement1>
  else
    if ( <cond-expr> )
      <statement2>
    else
      <statement3>
  ```
Nested if (multi-way branch)

general form

```plaintext
if (<cond-expr1> )
    <statement1>
else if (<cond-expr2> )
    <statement2>
else if ( <cond-expr3> )
    <statement3>
    ......
else
    <statementN>
```

Exactly one of the statements is executed

It is the first one for which expr$_i$ is true, or statementN if all expr$_i$ are false.
**if Efficiency Notes**

- avoid unnecessary comparisons (relational and logical operator expressions);
- work from most likely cases to least likely
- work from most general cases to most specific
Fix it:

```c
char sunny;
int temp;

cout << endl << "Is it sunny out (Y/N)"; cin >> sunny;

cout << "What's the temp outside"; cin >> temp;

if ( sunny == 'Y' || sunny == 'y' )
    if ( temp >= 70 )
        cout << "Head to the beach" << endl;
else
    cout << "Bring your umbrella" << endl;
```
Fix it:

```cpp
char sunny;
int temp;

cout << endl << "Is it sunny out (Y/N)?";
cin >> sunny;

cout << "What's the temp outside?";
cin >> temp;

if ( sunny == 'Y' || sunny == 'y' )
    if ( temp >= 70 )
        cout << "Head to the beach" << endl;
else
    cout << "Bring your umbrella" << endl;
```
char sunny;
int temp;

cout << endl << "Is it sunny out (Y/N)?";
cin >> sunny;

cout << "What’s the temp outside?";
cin >> temp;

if ( sunny == ‘Y’ || sunny == ‘y’ )
    if ( temp >= 70 )
        cout << "Head to the beach" << endl;
    else
        cout << "Bring your umbrella" << endl;
char sunny;
int temp;

cout << endl << "Is it sunny out (Y/N)?";
cin >> sunny;

cout << "What's the temp outside?";
cin >> temp;

if ( sunny == 'Y' || sunny == 'y' )
{
    if ( temp >= 70 )
        cout << "Head to the beach" << endl;
}
else
    cout << "Bring your umbrella" << endl;
if a logical expr is true, its complement is false, and vice versa
if we test an expr, we should not then test its complement (unnecessary)

```cpp
if (x < 5)
    cout << "x is less than 5";
else if (x >= 5)
    cout << "x is at least 5";
```
Complementing Expressions

• if a logical expr is true, its complement is false, and vice versa
• if we test an expr, we should not then test its complement (unnecessary)

if (x < 5)
    cout << "x is less than 5";
else // if (x >= 5) not needed
    cout << "x is at least 5";
Complementing (2)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>&gt;=</td>
</tr>
<tr>
<td>&lt;=</td>
<td>&gt;</td>
</tr>
<tr>
<td>==</td>
<td>!=</td>
</tr>
</tbody>
</table>
Complementing (2)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>&gt;=</td>
</tr>
<tr>
<td>&lt;=</td>
<td>&gt;</td>
</tr>
<tr>
<td>==</td>
<td>!=</td>
</tr>
</tbody>
</table>

Expr

<table>
<thead>
<tr>
<th>Expr</th>
<th>Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>x &lt; 5</td>
<td></td>
</tr>
</tbody>
</table>
## Complementing (2)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>&gt;=</td>
</tr>
<tr>
<td>&lt;=</td>
<td>&gt;</td>
</tr>
<tr>
<td>==</td>
<td>!=</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expr</th>
<th>Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>x &lt; 5</td>
<td>! (x &lt; 5)</td>
</tr>
</tbody>
</table>
## Complementing (2)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>&gt;=</td>
</tr>
<tr>
<td>&lt;=</td>
<td>&gt;</td>
</tr>
<tr>
<td>==</td>
<td>!=</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expr</th>
<th>Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>x &lt; 5</td>
<td>! (x &lt; 5)</td>
</tr>
<tr>
<td></td>
<td>x &gt;= 5</td>
</tr>
</tbody>
</table>
Complementing (3)

- DeMorgan's Theorem:
  - complement a compound logical expression by first writing the complement of each subexpression and then changing each AND to OR and each OR to AND.

complement

(18 <= age) && (age <= 22)  
\(\neg\) (18 > age)  \(\|\)  (age > 22)
Number Line Example

age < 18

18

18 <= age && age <= 22

22

age > 22
Number Line Example

- age < 18
- age > 22
- Not: 18 <= age <= 22
Short-Circuit Evaluation (AND)

- if left operand false
  - entire expression false
  - no point to evaluate right
- Example:

  \[(i == 5) \land \land (\sqrt{j} > 10)\]
Short-Circuit Evaluation (AND)

- if left operand false
  - entire expression false
  - no point to evaluate right
- Example:

\[(i == 5) \land\land (\sqrt{j} > 10)\]

- more efficient if right operand expensive
Short-Circuit Evaluation (AND)

• if left operand false
  – entire expression false
  – no point to evaluate right
• Example:

\[(i == 5) \land \land (\sqrt{j} > 10)\]

• more efficient if right operand expensive
• useful for error prevention

\[(i \neq 0) \land \land (j / i > 15)\]
Short-Circuit Evaluation (OR)

• if left operand true
  – entire expression true
  – no point to evaluate right
• again, promotes efficiency
int x;
cout << "Enter x: " << endl;
cin >> x; // Suggested inputs: 2, 4, 0

if (x <= 3)
  if (x != 0)
  {
      cout << "Hello from second if.\n";
  }
else
{
    cout << "Hello from else.\n";
}

cout << "End\n";
int x;
cout << "Enter x: " << endl;
cin >> x; // Suggested inputs: 2, 4, 0

if (x <= 3)
    if (x != 0)
    {
        cout << "Hello from second if.\n";
    }
else
    {
        cout << "Hello from else.\n";
    }

cout << "End\n";

What prints?
Questions?

- Relational and Logical Operators and Expressions
- Control Structures
- if and if-else statements
- nested if