Unit 5

Conditional ('if'...'else') Statements
Outline

• Comparison operators
• Conditional statements (3 forms)
  – if statements
  – if..else statements
  – if..else if..else statements
• Compound conditional statements
  – AND, OR, and NOT (&&, ||, !) operators
Control Structures

• We need ways of making **decisions** in our program
  – To repeat code until we want it to stop
  – To only execute certain code if a condition is true
  – To execute one segment of code or another

• Language constructs that allow us to make decisions are referred to as **control structures**

• The common ones are:
  – if statements
  – switch statements
  – while loops
  – for loops
Making Decisions

• **Comparison** of values results in 'true' or 'false' results

• Using **comparisons** we can develop **conditions**
  – If $x > 0 \text{ AND } y > 0$, take some action
  – If USC has 12 wins we will play for the championship

• Using **conditions** we can make **decisions** about what code to execute
Comparison Operators

• To perform comparison of variables, constants, or expressions in C/C++ we can use the basic 6 comparison operators

<table>
<thead>
<tr>
<th>Operator(s)</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equality</td>
<td>if(x == y)</td>
</tr>
<tr>
<td>!=</td>
<td>Inequality</td>
<td>if(x != 7)</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less-than</td>
<td>if(x &lt; 0)</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater-than</td>
<td>if(y &gt; x)</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less-than OR equal to</td>
<td>if(x &lt;= -3)</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater-than OR equal to</td>
<td>if(y &gt;= 2)</td>
</tr>
</tbody>
</table>
Conditional Execution – 'if'

- if statements are the primary structure we use to execute a block of code only if a certain condition is met (true).
- Skips code if condition is false

```
if (condition1)
{
    // executed if condition1 is true
    // following statements
    x = x + 1;
}
```

(T / F) = Condition

- True
- False

Diagram:
- If Block Statements
- Following statements
- True
- False
- condition 1

Flowchart:
- IF Block Statements
- Following statements
- True
- False
Conditional Execution – 'if..else'

- else statements are always **optional** and will execute when if conditions are **false**

(T / F) = Condition

1 1

if (condition1)
    // executed if condition1 is true
else
    // executed if condition1 above is false

3 3

// following statements

if Block Statements

else Block Statements

Following statements
What Goes In an if Block

• What do we put in an **if** or an **else** statement?
  – ANYTHING!
    – Expressions & variable assignment
    – Function calls
    – Even other if..else statements

```cpp
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
    double val;
    cout << "Enter a pos. #" << endl;
    cin >> val;

    if( val >= 0) {
        double res = sqrt(x);
        cout << "Result=" << res << endl;
    }
    else {
        cout << "Error!" << endl;
    }

    return 0;
}
```
else Options

- else block is optional
  - Can have 0 or 1
  - But no more than 1

```c
if (x > 0) {
    y = x;
}
```

No else block...OK

```c
if (x > 0) {
    y = x;
}
else
{
    x = -1;
}
```

2 or more else blocks...NOT OK!

```c
if (x > 0) {
    y = x;
}
else
{
    x = 0;
}
```

One else block...OK
else if Options

- else if blocks are always **optional** but you can use as many as you like
  - Can have 0, 1, 2, ..., N else if blocks

```plaintext
if (x > 0)
{
    y = x;
}
else if(y == 0)
{
    x = 0;
}
else if(y < 0)
{
    x = -y;
}
else
{
    x = -1;
}
```

No else if block..OK

```plaintext
if (x > 0)
{
    y = x;
}
else if(y == 0)
{
    x = y;
}
else if(y < 0)
{
    x = -y;
}
else
{
    x = -1;
}
```

Several else if blocks..OK
An Analogy for 'if' structures

• Always start with an 'if'

• Can have an 'else' but is not required

• Can have any number of 'else if' statements

• The next 'if' starts another independent if statement (both can execute)
Initial Exercises

• cpp/conditionals/extracredit
• cpp/conditionals/stoplight
• cpp/conditionals/stoplight3
• cpp/conditionals/nestedec
What Goes In an if Condition

• What do we put in an if condition?
  • ANYTHING.
    – The compiler will interpret what is in the parentheses as a Boolean
      • 0 = false
      • Non-0 = true

```c
int main()
{
    int x, y, val;
    bool quit;

    // Uses Boolean result of comparison
    if( x > 0 ) { /* code */ }

    // Uses value of bool variable.
    // Executes if quit == true.
    if( quit ) { /* code */ }

    // Interprets number as a bool
    // Executes if val is non-zero
    if( val ) { /* code */ }

    // Interprets return value as bool
    // Executes if the min is non-zero
    if( min(x,y) ) { /* code */ }

    return 0;
}
```
Mutually Exclusive Conditions

- What will each implementation print if 'grade' is 95?

```cpp
if (grade >= 90) {
    cout << "A range" << endl;
} else if (grade >= 80) {
    cout << "B range" << endl;
} else if (grade >= 70) {
    cout << "C range" << endl;
} else if (grade >= 60) {
    cout << "D range" << endl;
} else {
    cout << "Not gonna happen!" << endl;
}
```

```cpp
if (grade >= 90) {
    cout << "A range" << endl;
} else if (grade >= 80) {
    cout << "B range" << endl;
} else if (grade >= 70) {
    cout << "C range" << endl;
} else if (grade >= 60) {
    cout << "D range" << endl;
} else {
    cout << "Not gonna happen!" << endl;
}
```
If...Else If...Else

• Guideline:
  – If various blocks of code are mutually exclusive then put them in an if..else if..else structure and not many individual if statements

```cpp
if (grade >= 90)
{
    cout << "A range" << endl;
}
else if (grade >= 80)
{
    cout << "B range" << endl;
}
else if (grade >= 70)
{
    cout << "C range" << endl;
}
else if (grade >= 60)
{
    cout << "D range" << endl;
}
else
{
    cout << "Not gonna happen!" << endl;
}
```

Only 1 'cout' should execute. Use else if statements.
COMPOUND CONDITIONS AND LOGICAL OPERATORS
Logical Operators

• We can create compound conditions by using the logical AND, OR, and NOT operator

<table>
<thead>
<tr>
<th>Operator(s)</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>AND</td>
<td>if( (x==0) &amp;&amp; (y==0) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>NOT</td>
<td>if( !x )</td>
</tr>
</tbody>
</table>
Logical AND, OR, NOT

• The following tables show how the logical operations are evaluated under any set of values

• AND:
  – All inputs must be true for resulting expression to be true
  – If even one is false, the condition is fails (false)

• OR:
  – If any input is true the condition evaluates to true

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</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>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>
Order of Evaluation (Precedence)

• Precedence (Order of Operations)
  – Highest = ! (NOT)
  – Next = && (AND)
  – Lowest = || (OR)

• Better strategy:
  – Explicitly parenthesize everything

```c
int main()
{
    bool a, b, c;
    /* Some code that sets a, b, c */
    if( a && !b || c)
    {
        /* Some code */
    }
}
```
if statement Tips

- https://sourcemaking.com/refactoring/simplifying-conditional-expressions
Common Mistakes 1

• Using assignment operator ( = ) rather than equality check operator ( == )
  – If you accidentally use ' = ', it will convert the assigned value to a Boolean

• Using multiple if statements rather than if..else or if..else if statements
  – Without looking at the conditions, two 'if' statements imply both could be true while 'if..else' implies one or the other

```c
int main()
{
    int x, y;
    cin >> x >> y;

    // Wrong!
    if( x = 0 ) { /* some code */ }  // Right!
    if( x == 0 ) { /* some code */ }

    // Wrong!
    if( x != y ) { x = 5; }  
    if( x == y ) { y = 7; }

    // Right
    if( x != y ) { x = 5; }
    else       { y = 7; }
    return 0;
}
```
Common Mistakes 2

- All conditions must be formulated as a combination of comparisons of two values at a time.
Comparing Floating Point (Doubles) Numbers

- doubles effectively represent a number in binary with a technique similar to scientific notation \((+7.54\times10^5)\)
- However performing operations on doubles often leads to slight rounding errors
  - One result yields \(+7.541\times10^5\) while another yields \(+7.539\times10^5\)
- This makes comparison difficult
  - Thus we rarely check for exact equality
  - Instead, we take the different of the two numbers and take the absolute value (\(\text{abs}\)\) in \(<\text{cmath}\>\) and see if it is within a small epsilon

```cpp
#include <cmath>
using namespace std;

int main()
{
    double x, y;
    x = 1.0 / 10;
    y = 0.1;

    // Wrong!
    if( x == y )
    {
        /* some code */
    }

    // Right!
    if( abs(x-y) < 1e-6 )
    {
        /* some code */
    }

    return 0;
}
```
PROBLEM SOLVING IDIOMS
Rule/Exception Idiom

• **Name**: Rule/Exception
• **Description**: Perform a default action and then use an ‘if’ to correct for exceptional cases
• **Structure**: Default action code followed by if statement with code to correct the exceptional case
• **Example(s)**:
  – Shipping for "members"

```cpp
// Default action
if( /* Exceptional Case */ )
{
    // Code to apply to
    // exceptional case
}
```

```cpp
bool primeMember = /* set somehow */;
double shippingFee = 7.99;
if( primeMember == true )
{
    shippingFee = 0;
}
```
Rule Exception Idiom (2)

- **Connections**: Often equivalent to use the 'else' for one of the cases

```cpp
bool primeMember = /* set somehow */;

double shippingFee = 7.99;
if( primeMember == true )
{
    shippingFee = 0;
}
else
{
    shippingFee = 7.99;
}
```
Look-up Table Idiom

- **Name:** Look-up Table (Parallel cases)
- **Description:** Break input into mutually exclusive cases, taking some action or producing some output in each case
- **Structure:** Single level 'if..else if..else' statement

<table>
<thead>
<tr>
<th>Score (input)</th>
<th>Grade (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 90</td>
<td>A</td>
</tr>
<tr>
<td>80-89</td>
<td>B</td>
</tr>
<tr>
<td>70-79</td>
<td>C</td>
</tr>
<tr>
<td>55-69</td>
<td>D</td>
</tr>
<tr>
<td>&lt; 55</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Weather</th>
<th>Dress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hot</td>
<td>T-shirt</td>
</tr>
<tr>
<td>Mild</td>
<td>Long Sleeves</td>
</tr>
<tr>
<td>Cold</td>
<td>Sweater</td>
</tr>
</tbody>
</table>

```
if( /* Condition 1 */ )
{
    // Case 1 code
}
else if( /* Condition 2 */ )
{
    // Case 2 code
}
else if( /* Condition 3 */ )
{
    // Case 3 code
}
else {  /* Default */
    // Default code
}
```

```
if( weather == "hot" ) {
    clothing = "t-shirt";
}
else if( weather == "mild" ) {
    clothing = "long sleeves";
}
else {  /* Default */
    clothing = "sweater";
}
```
Look-up Table Idiom Examples (2)

- Example(s)

```cpp
if( level == "freshman" )
{
}
else if( level == "sophomore" )
{
}
else if( level == "junior" )
{
}
else if( level == "senior" )
{
}
else { /* Others/Errors */
}

int p1Score, p2Score;
/* Set variables */
if( p1Score > p2Score) {
    cout << "Player 1 wins";
}
else if(p1Score < p2Score){
    cout << "Player 2 wins";
}
else {
    cout << "Tie";
}
```
• **Connections:** The look-up table idiom (parallel cases) can be thought of as a structural OR'ing if the actions are the same in several cases
  - Underclassmen if freshman OR sophomore
  - Player 1 wins if Player 2 forfeits OR they score more points

```cpp
if( level == "freshman" || level == "sophomore" )
{
    cout << "Underclassmen";
}

if( level == "freshman" || level == "sophomore" )
{
    cout << "Underclassmen";
}
else if( level == "sophomore" ){
    cout << "Underclassmen";
}
else if( level == "junior" ) {
    cout << "Upperclassmen";
}
else if( level == "senior" ) {
    cout << "Upperclassmen;"
}

int p1Score, p2Score;
bool p2Forfeit;

/* Set variables */

if( p2Forfeit == true) {
    cout << "Player 1 wins";
}
else if(p1Score > p2Score){
    cout << "Player 1 wins";
}
```
Decision Tree (Subcase) Idiom

- **Name**: Subcase
- **Description**: Further divide one case into one or more subcases
- **Structure**: Nested 'if' statements

```c
if( /* Condition 1 */ )
{
    // Case 1 code
    if( /* Subcondition 1a */ )
    {
        // Subcase 1a code
    }
    else {
        // Subcase 1b code
    }
}
else if( /* Condition 2 */ )
{
    // Case 2 code
    if( /* Subcondition 2a */ )
    {
        // Subcase 2a code
    }
}
else if( /* Condition 3 */ )
{
    // Case 3 code
    if( /* Subcondition 3a */ )
    {
        // Subcase 3a code
    }
}
else {
    // Case 4 code
}
```
SubCase Idiom (2)

• Examples:

```java
if( option1 == "accounts" )
{
    // account related services
    if( option2 == "balance" ) {
        // Code to retrieve balance
    }
    else if(option2 == "cancel" ) {
        // Code to cancel account
    }
} else if (option1 == "hours")
{
    // code to list hours
}
else
{
    // repeat options
}
```

```java
if( category == "manager" ) {
    bonus = .1*salary; // 10% bonus
    if( title == "ceo" ) {
        bonus += 1000; // 1000 extra
    }
} else {
    bonus = .05*salary; // 5% bonus
}
```
SubCase Idiom (3)

• **Connections:** Subcase can be thought of as a structural AND'ing
  – You have to be a MANAGER and a CEO to get the bonus
Exercises

• cpp/conditionals/taxbrackets
• cpp/conditionals/instock
• cpp/conditionals/bill2law
• cpp/conditionals/rps