

## 06 - Making Decisions - Part 2

Dr. Robert Lowe

Division of Mathematics and Computer Science  
Maryville College

# Outline

- 1 Some General Advice
- 2 Advanced Decision Making

# Git Work Sessions

- Always begin each work session with:

```
git pull
```

- Frequently commit!

```
git add -A
```

```
git commit -a
```

- When you do a commit, git will open nano for you to edit your messages. You can avoid opening the editor by using the `-m` option to specify a log message directly on the command line:

```
git commit -a -m 'log message here'
```

- Always make sure to commit and then push all changes at the end of a work session.

```
git push
```

# Programming Affirmations

- Do not be afraid to fail.
- Fail quickly, fail often.
- You are not the one who is behind. Pretty much everyone in this room feels like they are the only one who hasn't caught on.
- You are doing far better than you realize. Learning to code is not easy. That you are still here means you can do this!
- Programming is a repeated effort. It is full of false starts and scrapped efforts.
- If you are stuck, more code is rarely the answer. Instead go back to your design notes and try to find what you missed.
- Seeing a program through from beginning to end without backtracking and reworking almost never happens.

# Working With the Compiler

- The compiler generates two kinds of messages (warnings and errors).
- A **warning** is something that can indicate code that is suspected to be faulty.
- When the compiler issues a warning, it still compiles the program.
- An **error** is something that means the compiler cannot follow the meaning of the code. (Malformed syntax, invalid keywords, wrong types, etc.)
- When an error occurs, the compiler does not generate code.

# Locating Errors

```
even-odd.cpp: In function 'int main()':  
even-odd.cpp:19:5: error: expected ';' before '}' t  
    } else {  
    ^
```

- Compiler error messages will indicate where the error/warning was located.
- The format is `filename:line:column`
- The above error is from file `even-odd.cpp` line 19 column 5
- The location is where the problem was noticed. Not necessarily where it actually needs to be fixed.
- Compilers do nothing to detect logic errors!

## Challenge: Fix `proportion.cpp`

- 1 Make the directory `labs/week4`
- 2 Copy the file  
`examples/07-Decisions/proportion.cpp` to your  
`labs/week4` directory.
- 3 Try to compile `proportion.cpp`.
- 4 Use the compiler error messages to locate and fix the  
compiler errors.
- 5 Test the program. Fix any logic errors you may find.

# UNIX Tips and Shortcuts

- Typing part of a filename followed by the `tab` key will complete the filename for you.
- You can scroll through your command history by pressing up and down on the cursor keys.
- Repeat a selected command by pressing `enter`.
- You can repeat a command by pattern matching using `!`.  
For example, to repeat your last compiler line:  
    `!g++`  
    or  
    `!g`
- Try using these as you use the command line. More speed tips will follow.



# Testing for a Range of Values

- In your `examples/06-Decisions` folder, you will find `range.cpp`
- Run and test this program. Does it work?
- What is going on here?

```
int main()
{
    int num;

    //get a number
    cout << "Enter a number" << endl;
    cin >> num;

    //test to see if it is between 1 and 5
    if(1 <= num <= 5) {
        cout << "The number is between 1 and 5" << endl;
    } else {
        cout << "The number is not between 1 and 5" << endl;
    }
}
```

# Combinational Operators

**and**

a	b	a and b
F	F	F
F	T	F
T	F	F
T	T	T

**or**

a	b	a or b
F	F	F
F	T	T
T	F	T
T	T	T

**not**

a	not a
F	T
T	F

# Operator Precedence (Thus Far)

Operator	Description	Associativity
not, !	Logical Not	Left-to-Right
a*b, a/b, a%b	Multiply, Divide, Modulus	Left-to-Right
a+b, a-b	Addition and Subtraction	Left-to-Right
« , »	Insertion and Extraction	Left-to-Right
<, <= >, >=	Relational Operators	Left-to-Right
==, !=	Equality Operators	Left-to-Right
and, &&	Logical And	Left-to-Right
or,	Logical Or	Left-to-Right
=, +=, -= *=, /= %=	Assignment and Assignment	Right-to-Left

## Example: Range Validate

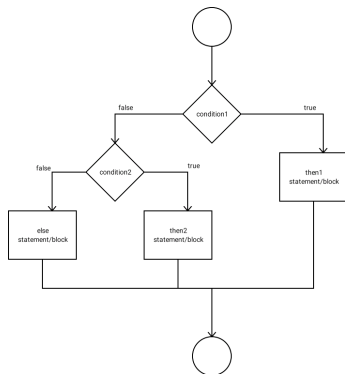
```
num >= 1 and num <= 5
```

- The above expression is the correct way to detect over a range.
- Copy `range.cpp` to your `labs/week4` folder and correct it.
- Make sure the program works!

# Multi-Way Branching: If-Then-Else-If

```
if ( condition )  
    then statement/block  
else if ( condition )  
    then statement/block  
else  
    else statement/block
```

- The first *then statement/block* with a true condition executes.
- If no matches are found, the (optional) *else statement/block* executes.



## Example Snippet: Rock, Paper Scissors

```
if(player == 1) {  
    cout << "Rock" << endl;  
} else if(player == 2) {  
    cout << "Paper" << endl;  
} else if(PLAYER ==3) {  
    cout << "Scissors" << endl;  
}
```

# Challenge: The Stock Menu

Stock Portfolio Management System

Please Make a Selection

- 1 -- Buy a Stock
- 2 -- Sell a Stock
- 3 -- Report Current Holdings
- 4 -- Report Gains and Losses
- 5 -- Remove a Current Holding
- 6 -- Done! (quit)

Choice?

- ❶ Copy your `stock.cpp` file from your `labs/week2` directory to your `labs/week4` directory.
- ❷ Add logic so that it prints your menu selection. For instance, if you enter “1”, your program should reply with “Buy a Stock”
- ❸ Add logic so that if you select anything other than 1 through 6, your program displays an error message.