Variables and Operators in C++

CS 16: Solving Problems with Computers I
Lecture #3_PreRecorded Video

Ziad Matni
Dept. of Computer Science, UCSB

Part 1 of 2

Lecture Outline

Video Part 1

- More on Variables and Operations in C++

Video Part 2

- A Brief Unix Commands Primer
- An Introduction to GitHub

Characters in C++

char: single character

- Can be any single character from the keyboard
- To declare a variable of type char:

```
char letter;
```

- Character constants are enclosed in single quotes

```
char letter = 'a';
```

Strings in C++

`string`: a collection of characters (a *string* of characters)

- **string** is a ***class***, different from the primitive data types discussed so far.
  - We'll discuss classes further in the course

- Using C++ strings requires you to include the "string" module:

```
#include <string>
```

- To declare a variable of type string:

```
string name = "Homer Simpson";
```

- There are "older" types of strings called ***C-Strings*** that are still in use in C++
  - More on those later…

Note on ‘ vs “

- Single quotes are only used for **char** types
- Double quotes are only used for **string** types

- So, which of these is ok and which isn’t?

```
char letter1 = “a”;          X
char letter2 = ‘b’;          ✓
string town1 = “Mayberry”;   ✓
string town2 = ‘Xanadu’;     X
```

Type Compatibilities

- General Rule: **You cannot operate on differently typed variables.**
  - Except with **int** and **double** types
  - Just like in most computer languages
- So, if: then:

```
int my_var = 2;
char my_char = 'x';
```

```
my_var + my_char
```

is a syntax error

- There are rules with operations between **int** and **double**…

int ←→ double

- Variables of type *double* should **not** be assigned to variables of type *int*
- Variable of type *int*, however, **can** normally be stored in variables of type *double*
  - The compiler will understand this and convert the int into a double automatically

EXAMPLE:
```
double numero;
numero = 2;
```

- *numero* will contain **2.0000** (w/ an unfixed number of places after decimal pt)

EXAMPLE:
```
int numero;
numero = 2.789;
```

- *numero* will contain **2** (because it's defined as an integer)

Declaring Constants in C++

- You can declare a variable to be a constant for the entire program
  - Useful for mathematical constants and other things
- Constants cannot be changed, once declared and initialized
- You have to use the **`const`** keyword
- If you want the constant to be valid throughout the program, you have to declare it **outside** of main() function.

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

const double GACC = 9.81;

int main() {
    double mass, weight;
    cout << "Enter mass: ";
    cin >> mass;
    weight = mass * GACC;

    cout << "The weight (on Earth) is:";
    cout << weight << endl;

    return 0;
}
```

Quick Note on NewLine Characters

- To print a newline character, you can use either:
  - "\n" a string with the newline character in it
  - '\n' a single character (not commonly used)
  - endl "end line" – comes with <iostream> (std::endl)

- Examples:

```
cout << "...and then he ate it!/n";

cout << "Your age is: " << age << endl;

cout << "/n/nWow!/nWhee!/n";
```

Quick Note on Escape Characters

- To print something explicitly, use the escape character ('\')

- For example:

```
cout << "And I said, \"Hi!\"";
```

Will print out:

```
And I said, "Hi!"
```

More here!
https://en.cppreference.com/w/cpp/language/escape

Variable Comparisons

- When variables are being ***compared*** to one another, we use ***different symbols***
- a is equal to b — a == b
- a is not equal to b — a != b
- a is larger than b — a > b
- a is larger than or equal to b — a >= b
- a is smaller that b — a < b
- a is smaller than or equal to b — a <= b

Note:
The outcome of these comparisons are always either true or false

i.e. Boolean

Boolean variables:
false = 0
true ≠ 0
(note lower-case!!!)

== is NOT THE SAME AS =

- The operator = is an *assignment* operator
  - Assigns a VALUE to a VARIABLE
  - e.g. `letter = 'm';`

- The operator == is a *comparison* operator
  - Compares if two things are EQUAL and returns a Boolean 'true' if so
  - e.g.

```
int number = 7;
bool isIt = (number == 7);     // isIt = true
```

Booleans in C++

`bool`: a binary value of either “true” (1) or “false” (0).

- C++ allows the interchange between true = 1 and false = 0
- You can perform LOGICAL operations on this type (like logic AND, logic OR, etc…)
  - These have specific operators (next slide)

Review of Boolean Expressions: *AND, OR, NOT*

AND operator &&

- (expression 1) && (expression 2)
- Outcome is **true** if ***both*** expressions are true (otherwise it's false)

OR operator ||

- (expression 1) || (expression 2)
- Outcome is **true** if ***either*** expression is true

Note: no space between each '|' character!

NOT operator !

- !(expression)
- False, if the expression is True (and vice versa)

Truth Tables for Boolean Operations

AND

X	Y	X && Y
F	F	F
F	T	F
T	F	F
T	T	T

OR

X	Y	X || Y
F	F	F
F	T	T
T	F	T
T	T	T

NOT

X	! X
F	T
T	F

IMPORTANT NOTES:

1. AND and OR are **not opposites** of each other!!
2. AND: if *just one* condition is false, then the outcome is false
3. OR: if *at least one* condition is true, then the outcome is true
4. AND and OR are **commutative, but not when mixed** (so, order matters)

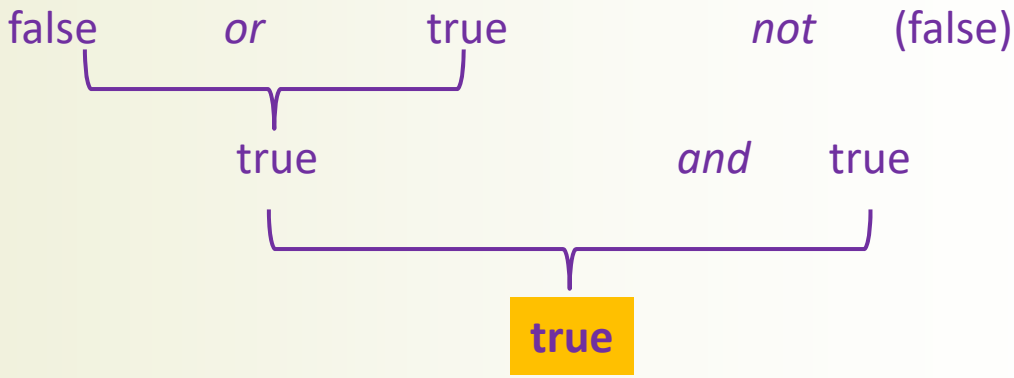
   X && Y = Y && X

   X && (Y || Z) is not the same as (X && Y) || Z

Example

- What is the Boolean value of **`outcome`** in this scenario?

```
int a = 5, b = 6, c = 7;
bool outcome = ( ((a >=b ) || (c – b < a)) && !(a + b < c) );
```



Arithmetic Expressions

- Precedence rules for operators are the same as what you used in your algebra classes
  - EXAMPLE: `x + y * z` (y is multiplied by z first)
- Use parentheses to force the order of operations (recommended)
  - EXAMPLE: `(x + y) * z` (x and y are added first)

Assignment vs. **Algebraic** Statements

- C++ syntax is NOT the same as in Algebra!
  - Applies to almost all programming languages…

EXAMPLE:

number = number + 3

In C++, it means:

- take the *current* value of "number",
- add 3 to it,
- then reassign that *new value* to the variable "number"

C++ shortcut:

number += 3

Also works with:

-= *= /= %= *etc…*

Operator Shorthand

- Some expressions occur so often that C++ contains shorthand operators for them

- All arithmetic operators can be used this way:

  - **count = count + 2;** ---can be written as--- **count += 2;**
  - **bonus = bonus * 2;** ---can be written as--- **bonus *= 2;**
  - **time = time / factor;** ---can be written as--- **time /= factor;**
  - **remainder = remainder % (cnt1+ cnt2);**
    ---can be written as--- **remainder %= (cnt1 + cnt2);**

Incrementing/Decrementing by 1

In C++, you can use the following shorthands:

- Instead of saying `count = count + 1;`

  You can say `count++;`

  You can ALSO say `++count;`

  There IS a difference between these 2 expressions (more on that later)

- Likewise, instead of `count = count - 1;`

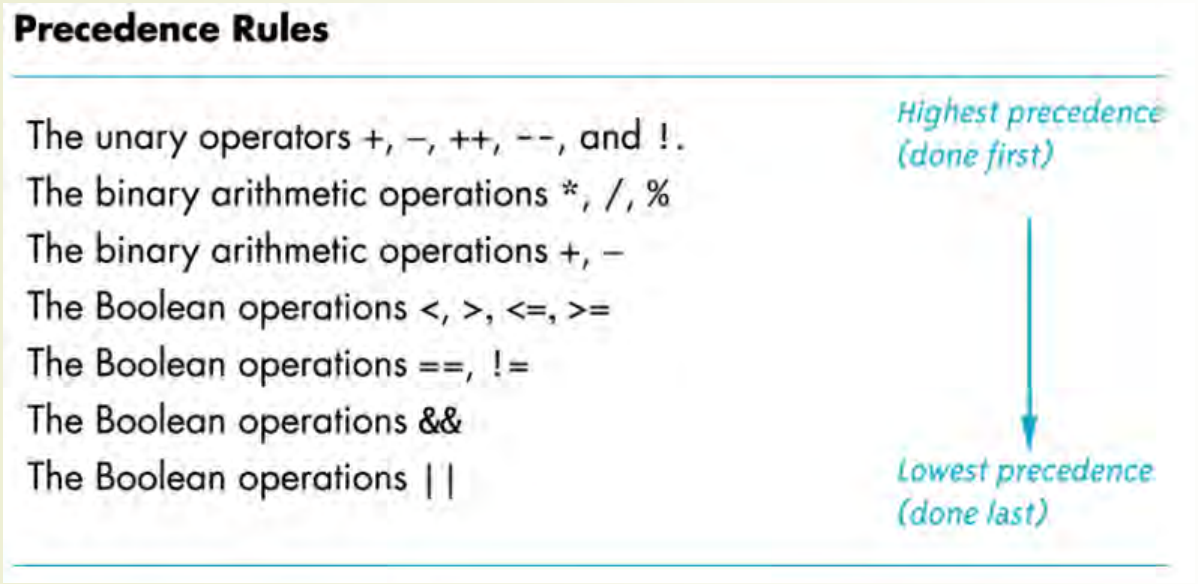
  You can say `count--;`

  You can ALSO say `--count;`

  Again, there IS a difference between these 2 expressions

Precedence Rules on Operations in C++

- If parenthesis are omitted from C++ expressions, the default precedence of operations is:



END OF PART 1

A Brief Unix and Gi

CS 16: Solving Problems with C
Lecture #3_PreRecorded

Ziad Matni
Dept. of Computer Science

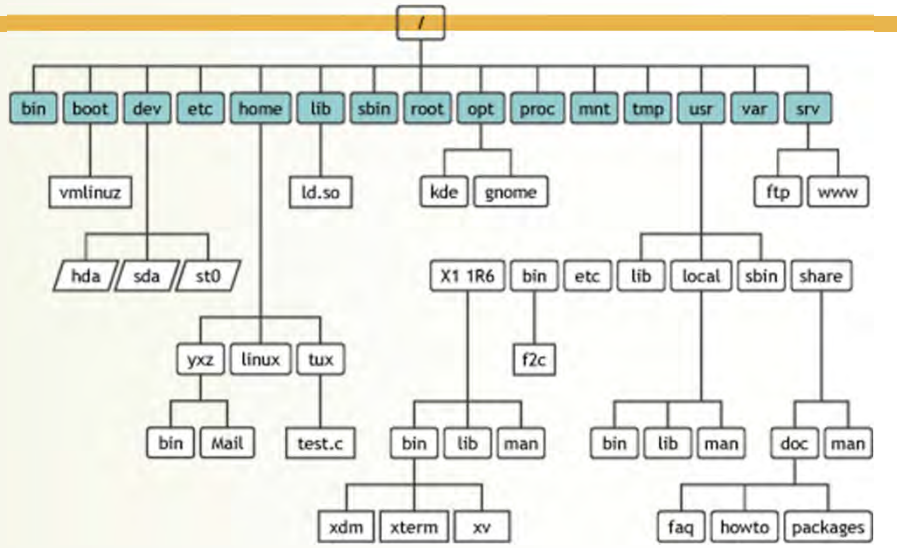
```cpp
int main(int argc, char *argv[])
{
    if (argc > 1)
        filename = argv[1];
    ifstream setIn(filename);
    ifstream vecIn(filename);
    set<string> wordSet = getWordSet(setIn);
    vector<string> wordVec = getWordVec(vecIn);
    map<string, string> wordMap = generateMap(wordVec);

    string name = filename.substr(0, filename.size() - 4);
    string setFilename = name + "_set.txt";
    string vecFilename = name + "_vec.txt";
    string mapFilename = name + "_1_1.txt";

    // Writes set file
    ofstream setOut(setFilename);
    for (set<string>::iterator it = wordSet.begin(); it != wordSet.end(); it++)
    {
        setOut << *it << endl;
    }
    setOut.close();

    // Writes vector file
    ofstream vecOut(vecFilename);
    for (int i = 0; i < wordVec.size(); ++i)
    {
        vecOut << wordVec[i] << endl;
    }
    vecOut.close();

    // Writes to map
    ofstream mapOut(mapFilename);
    printMap(wordMap, mapOut);
    mapOut.close();

    // Generate and print random string
    string str = "";
    for (int i = 0; i < 100; i++)
    {
        cout << wordMap[str] << " ";
        str = wordMap[str];
    }
    cout << endl << endl << endl;

    // Generate more intelligent map
    map<string, vector<string>> wordVecMap;
    str = "";
    for (int i = 0; i < wordVec.size(); i++)
    {
        wordVecMap[str].push_back(wordVec[i]);
        str = wordVec[i];
    }
```

Part 2 of 2

Directory Structure

- Directories (also called folders) are constructs where multiple files (or other directories) can be located.

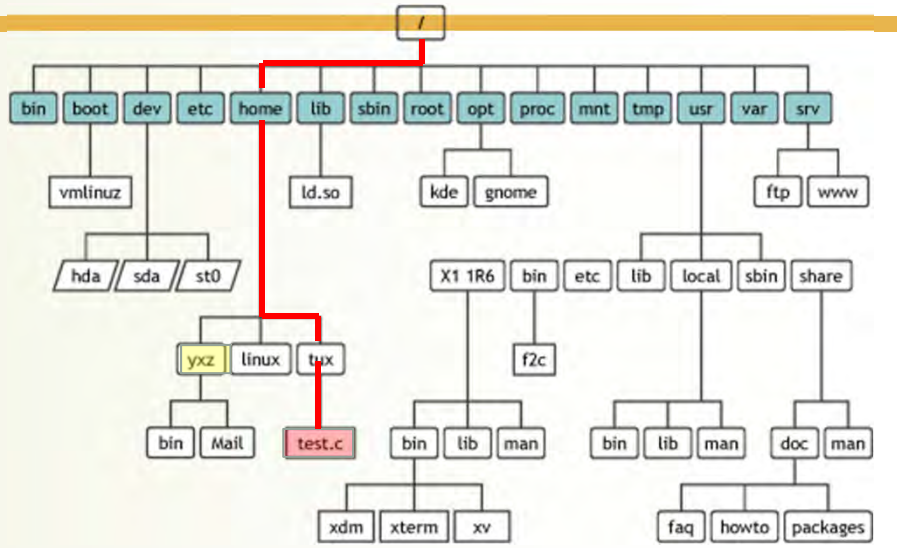
- The very “top” directory is called the “root” directory



Pathnames

- Pathnames refer to the **location** of a file or directory

- If pathnames "spell out" the entire "path", starting from the root directory, we call that an ***absolute pathname***
  - Example: **/home/tux/test.c**



- **Relative pathnames** are those that describe a location relative to another location
  - Example: If you're in the yxz directory (in yellow), then you can describe test.c as being at: **../tux/test.c** because the ".." means "one directory above"

Pathname Shorthands

- The following are common pathname shorthands in Unix:

```
~    home directory
..   The directory above this one
.    The current directory
```

Linux File Management Commands

- **`cd`** change directory, e.g. `cd /home/tux/test.c`
- **`pwd`** show me this directory, e.g. `pwd`
- **`ls`** list directory, e.g. `ls`
- **`cp`** copy file, e.g. `cp orig_file new_file`
  - If "new_file" existed before, it's now over-written
- **`rm`** remove (i.e. delete) file, e.g. `rm filename`
- **`mv`** move (i.e. rename) file, e.g. `mv orig_file new_file`

Linux Text File Manipulation Commands

- **`cat`** show contents of text file, e.g. `cat filename`
- **`more`** show contents of text file, but one page at a time, e.g. `more filename`
- **`head`** show contents of text file, but only the first few lines, e.g. `head filename`
- **`tail`** show contents of text file, but only the *last* few lines, e.g. `tail filename`

- **`man`** show manual/help page for a Linux command, e.g. `man more`

Linux Built-in Text Editors

- vim
- emacs

- Used differently from each other
  - Usually people pick one or the other to work with

Remote Machines

- You can connect to a remote machine, as demo'd in the lab
  - Using **ssh**

- The syntax is:

```
ssh username@machineInternetAddress
```

e.g.

```
ssh gauchos@csil.cs.ucsb.eu
```

Remote Machines

- But what if you want to copy files over from a remote machine to your "local" machine??
  - Or the other way around??

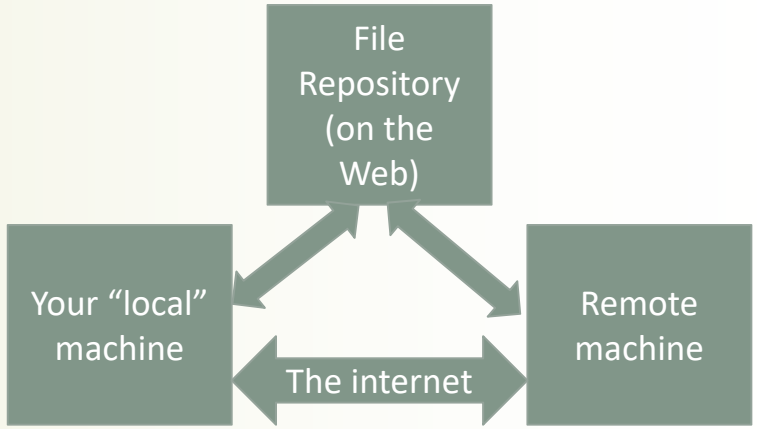
- *Several Ways To Do This:*

1. Use **scp** in Unix/Linux
   - Real programmers use Unix! ;)

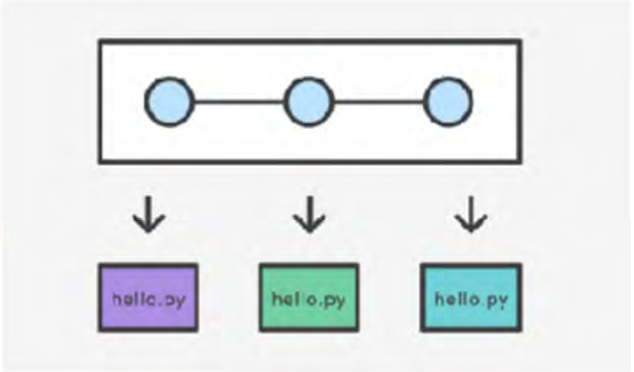
2. Use a program like **FileZilla** (Win10, Mac) or **CyberDuck** (Mac)
   - Free, GUI-based, and not hard to use

Remote Machines (Repositories)

- Another way is to make a repository (i.e. a place to keep your files) online using a service like **GitHub**



Introduction to **git**



- Git is a version control system (VCS).
- A VCS allows you to keep track of changes in a file (or groups of files) over time.

- Git allows you to store code on different computers and keep all these different "copies" in sync.
  - This is mostly the function that we want to use in this class

- Git also allows you to keep old versions of your code, in case you need to access it again.

Why Are We Learning **Git** in This Class?

- You will be using git in many CS classes to:
  - Collaborate with others
  - Share code ownership with TAs/instructors
  - Work on larger projects
  - Provide feedback on work in progress
- This is also about learning a very commonly used professional software development tool
  - I can guarantee that you will make use of this in your CS careers…

Git Concepts

- **repo** (short for repository): a place where all your code and its history is stored in one main file folder
- **Cloning** a repo: making a copy of a repo on another machine
  (you can **fork** a project first, but we won't be doing that in CS16)
- **Commit** a change: when you make a change, you "solidify" it by committing it
- **Push** the changes: after committing, you update your repo using a "push"
- Request a **pull**: getting the most up-to-date repo on your local machine

The Workflow of an Existing Git

This assumes that you've already cloned your repo on your machine

- On UNIX, use `git clone <the address of the online repo>`

1. Make changes to existing file(s)
2. If you create a NEW file, you have to add it in git
   - On UNIX, use `git add <filename>`
3. Commit changes
   - On Unix, use `git commit -m "comments"`
4. Push changes
   - On Unix, use `git push`

We will practice this in the new lab next week!

Where are we Going to Make Repos???

- On **GitHub** https://www.github.com

- The most popular git service used by of coders everywhere
  - As of Jan. 2020, it has 40 million users and >190 million repos
  - Now owned by Microsoft (woohoo?)

- We'll cover its use in CS16 further in **lab02** next week!

</LECTURE>