*Notes 10/4*

Divide by 0 errors in C++ are not standardized – compilers writers can do whatever they want. Many compilers won’t check, they’ll just give the commands to the CPU, so your program will either crash or create very strange results (most computers will crash.)

As with all undefined behaviors, you should avoid dividing by zero because you really don’t know what will happen – it’s all up to the compiler. Fuck the compiler! Your program should do what *you* want it to

Another undefined behavior is trying to represent a value greater than ~2billion with an int. Usually it will truncate all numbers except the first 32 bits (the first couple digits) so if you try to multiply huge values, you will get some weird value and no error will be triggered. If you utilize big numbers, use double instead of int to avoid undefined behavior.

// = comment. Compiler doesn’t read it, it’s just for human comprehension.

A string variable is just like int and double, except it is a series of characters (letters, digits, punctuation, spaces, newline) instead of numerical values. If you want to declare a string in your program, you must put *#include <string>* at the start of your program.

getline is a library function that you use like this:  
*getline(cin, personsName)*  
How is getline different from cin? Cin starts reading when it gets to the first non-space character and *stops* when it reaches a space again. Getline starts reading at the first non-space character and *keeps reading until it reaches a newline character (almost always the end of the input, when the user presses enter).*

Whenever you collect string input, you should use getline rather than cin because cin will *truncate* any string values with more than one word.

Syntax is *getline(cin, variable to be initialized)*

An obvious symptom of misuse of cin vs getline is when your program races ahead and disregards your input after typing the first string – this is because subsequent cin’s are reading the second and third words in the initial input instead of waiting for the next prompt.

Cin.ignore is a function you can use to restrict cin input. For example:  
*Cin.ignore(100000, ‘\n’)*  
tells the computer to execute cin until it has consumed 100000 characters or it reaches a newline character. The first number is usually made large enough to be irrelevant (no one will type 100000 characters.)

**Use cin.ignore when you need to read an integer/double with cin and the very next thing you need to read with cin is a string.**

You can use special characters in strings to make tabs, newlines, etc

**\t = tab**. Example:  
*cout << “ab\tc”*  
would display   
“ab c”

**\n = newline**. Example  
*cout << “ab\nc”*would display   
“ab  
c”

**\” = quotes.** Example  
*cout << “He said, \”Help!\””*would display  
*“He said, “Help!””*

**\\ = backslash.** Example:  
*cout << “Male\\female”*would display  
“Male\female”

If you use double quotes, you refer to a character string as a whole; single quotes refer to a single character as a type. If you refer to ‘\n’ you are talking about the newline character; if you refer to “\n” you are talking about the literal character string \n.

// is for comments; use them like you would use a subheader in a newpaper. Don’t overuse comments or your code will be harder, not easier, to read, and you’ll sound like pedantic asshole.

If statements can be used to check conditions and execute different commands depending on the condition.

Syntax is:  
*if (condition)  
 statementIfTrue  
else  
 statementIfFalse*You can toss *and* in the condition to utilize multiple conditions for your if statement. Use curly braces to do lots of things in the *statementIfTrue* section.