Website: http://cs.ucla.edu/classes/fall19/cs31
cs31.seas.ucla.edu

Midterm: 11/20, 6:15pm - 7:50pm in Moore 100

If your correctness score is below 50, it may not be because of a lack of understanding of C++, but something more fundamental: You ignored repeated admonitions in the spec and in class to avoid specific foolish mistakes, yet you made them anyway. Whatever your field of study is, you must fix this characteristic about yourself. No employer would dare hire someone who ignores repeated spoken and written directives: You'd pose a risk to the safety of yourself and others if you ignore safety rules, a risk to the financial health of the company if you ignore legal regulations, and a drain on productivity if your ignoring specifications causes you or others to devote more time later on to correct your mistakes.

Final: 12/7, 11:30am - 2:30pm

scp Desktop/vampires.cpp charlesx@cs31.seas.ucla.edu:Desktop
ssh charlesx@cs31.seas.ucla.edu
cp Desktop/vampires.cpp .
g31 -o x vampires.cpp

9/27: History of Computers

- Jacquer (loom), Da Vinci (reprogrammable vehicle), Babitch (difference engine) 1830s, 1880 U.S. Census/Hollerith --> IBM, (tabulating machine), WWII Harvard MK. I for ballistics + ENIAC, SWAC at UCLA + SEAC
    - ENIAC 6 programmers: Kathleen McNulty, Frances Bilas, Betty Jennings, Ruth Lichterman, Elizabeth Snyder, Marlyn Wescoff
- Early computers only used by businesses and gov't, predicted to become bigger, extremely fast development
- CPU- Central Processing Unit
- Memory- Stores values for later
    - Store and retrieve integers
    - Need to find ways to represent decimals, characters, etc.
- I/O Devices- Get data in and out
    - Changes with times/purpose
    - Input: keyboard, mouse, temperature reader, etc.
    - Output: actuators, etc.
- Standardization occurs finally in 1960s, takes time due to patents, ASCII mapping for US

9/30: Basics 1

- Always be testing under g++
- Back up work on USB
- Start stuff early
- Develop incrementally
- Programs written in a machine language
    - CPU: Accumulator, instruction counter
    - Arithmetic logic unit: computes calculations
    - Indicated memory address, operation code
        - Operation code splits memory item into 2 pieces

10/2: Basics 2

- Machine language example (takes positions 4 and 5, adds them, puts the answer into 6):
    - Memory: slower, bulk of memory in machine
        - 0: 21004
        - 1: 11005
        - 2: 22006
        - 3: 99999
        - 4: 00042
        - 5: 00013
        - 6: 00055
        -
        - 999: 82016
    - CPU: splits into 2, smaller amounts of memory, faster due to need to execute instructions
        - Accumulator: 00055
        - Instruction Counter: 003
        - 99: operation code, 999: address
            - 11: add the number at the indicated memory address to the accumulator
            - 21: copy the number at the indicated memory address into the accumulator
            - 22: copy the number in the accumulator into the indicated memory address
            - 99: halt
- Jean von Neumann- same number that holds numbers can also hold instructions
- Faster → Slower: CPU, memory, external drives
- Once something is stored in a location, the previous object becomes not available
- Assembly Language- A language that can be translated into a corresponding machine language program, is assembled by the assembler
- Vendor Lock-in: due to the difference in machine languages across brands, people tend to stick with one brand as machines improve, old
- FORTRAN-1957
    - Integer price = 42
    - Integer fees = 13
    - Integer total = price + fees
- Higher-level language
    - A program is compiled by the compiler
- C-1971, C++ - 1979/1980, Objective-C - kept alive by Apple, Java, C# - based on Java, Microsoft, Swift - Apple, Python, Perl, Ruby, Javascript
- C++ - designed by Bjarne Stroustrup in ~1980, widespread in 1985
    - Standardization efforts start, released C++ standard in 1998 by ISO (C++98)
    - 2011- revision of C++ Standard (C++11)
    - 2014 - revision (C++14)

    - 2017 - revision (C++17)
- Computers execute machine language programs, C++ is translated into machine language
    - Errors can occur in both phases
        - Syntax/compilation error - user misusing the language, cannot be translated by compiler
        - Logic/runtime error - correct use of language, incorrect logic

10/4: Discussion 1

- Bucket problem
    - 1: Fill the 5g
    - 2: Use the 5g to fill the 3g
    - 3: Empty the 3g
    - 4: Pour the rest of the 5g into the 3g
    - 5: Fill the 5g
    - 6: Pour the 5g into the 3g
- Roomba problem
    - Check dock
        - while (detectDock == false)
            - Get list of valid moves, with length i
            - Generate random # from (0, i - 1)
            - Move in direction of list index
        - end
- Many times logic errors resultant of forgetting an edge case
    - Extrema
- Testing
    - Run the code
    - Normal Cases
    - Edge Cases
- Tips
    - Separate project for each file
    - Test on SEASnet

10/7: Basics 3

- Spacing is irrelevant for most major programming languages (except Python)
- Case sensitive
- **#include <iostream>**
- **using namespace std;**
- **int main() {**
    - **cout << "Hello!" << endl**
- **}**
- Can use multiple namespaces, unless it would makes a call ambiguous → compilation error
- **#include <iostream>**
- **using namespace std;**
- **int main() {**
    - **cout << "How many hours did you work? ";**
    - **double hoursWorked;** // declaration → type identifier; use an underscore or case change for 2 word identifiers
    - **cin >> hoursWorked;**
    - **cout << "What is your hourly rate of pay? ";**
    - **double payRate;**
    - **cin >> payRate;**
    - **double amtEarned = hoursWorked * payRate;** // declaration → type identifier = expression;
    - **cout.setf(ios::fixed);** // sets to fixed point notation, not scientific notation, stays until another one is set
    - **cout.precision(2);** // sets to 2 digits past the decimal, rounds as necessary, doesn't change the value, only the display, stays unless another one is set
    - **cout << "You earned $" << (amtEarned) << endl;**
    - **cout << "$" << (0.1 * amtEarned) << " will be withheld." << endl;**
- **}**
- double: + or -, 10^-308 to 10^308, 15 or 16 significant digits
- int: from ~-2 billion to ~2 billion
- Undeclared variables not given a default value in C++
- Arithmetic expression: *, /, +, -, %, uses PEMDAS
- 14.3 / 5.0 → 2.86
- 14.3 / 5 → 2.86
- 14 / 5.0 → 2.8
    - If any operand is a double, the result will be a double
- 14 / 5 → 2
    - If all operands are int, the result will be an int, cuts off the decimal
- 14 % 5 → 4 (mod), same precedence as multiplication/division
- **#include <iostream>**
- **#include <string>**
- **using namespace std;**

- **int main() {**
    - **cout << “What is your name? “;**
    - **string personsName;**
    - **getline(cin, personsName);** // only usable for strings, reads the entire input line and is assigned to the variable
    - **cout << “What is your quest? “;**
    - **string quest;**
    - **getline(cin, quest);**
    - **cout << “Hello, brave “ << personsName << “!” << endl**
    - **cout << “You want “ << quest << endl;**
- **}**

10/9: Control Flow 1

- Division by 0
  - **int a = 10;**
  - **int b = a * a;**
  - **int c = 25 / (b - 100);**
- Undeclared variable
  - **double d;**
  - **double e = 2 * d;**
  - **cout << e;**
- Overflow
  - **int f = 1000;**
  - **int g = f * f * f;**
  - **int h = f * g;**
- **int main() {**
  - **cout << "What is your name? ";**
  - **string personsName;**
  - **getline(cin, personsName);** // only usable for strings, reads the entire input line and is assigned to the variable
  - **cout << "How old are you? ";**
  - **int age;**
  - **cin >> age;**
  - **cin.ignore(1234567, '\n')** // fix to problem below, 1st argument: # of characters to throw away, 2nd argument: character to throw away until, '\n' = new line
  - **cout << "What is your quest? ";**
  - **string quest;**
  - **getline(cin, quest);**
  - **cout << "Hello, brave " << personsName << "!" << endl;**
  - **cout << "You want " << quest << endl;**
  - **cout << "If you live, next year you will be " << age + 1 << endl;**
- **}**
- Input only consumes as much as it needs to satisfy the request
  - **string s;**
  - **int i;**
  - **getline(cin, s);**
  - **cin >> i;**
  - **getline(cin, s)**
    - Type X, type Y, backspace, type Z, type enter
    - OS writes: X, Y, BS, Space, BS, Z, CR LF (Character Return, Line Feed)
    - OS holds: XY, X, XZ
    - Available to program: X Z new line
      - Does not make immediately available to program to allow for error correction, sent to program when enter is pressed
  - In above main(), the cin consumes the input 32, but not the "new line"

        - The next cin consumes the “new line”, causing there to be no chance to input a string for “quest”
        - Due to the way cin works with a numeric data type; stops when it hits a non-integer
            - Therefore, this problem occurs when you read a number and then read a string using getline
            - Need to throw away what’s left in the input
- **int main() {**
    - **cout << “How many hours did you work? ”;**
    - **double hoursWorked;** // declaration → type identifier; use an underscore or case change for 2 word identifiers
    - **cin >> hoursWorked;**
    - **cout << “What is your hourly rate of pay? ”;**
    - **double payRate;**
    - **cin >> payRate;**
    - **double amtEarned = hoursWorked * payRate;** // declaration → type identifier = expression;
    - **cout.setf(ios::fixed);** // sets to fixed point notation, not scientific notation, stays until another one is set
    - **cout.precision(2);** // sets to 2 digits past the decimal, rounds as necessary, doesn’t change the value, only the display, stays until another one is set
    - **cout << “You earned $” << (amtEarned) << endl;**
      double withholdingRate;
    - **if (payRate >= 14) {**
        - **withholdingRate = 0.1;**
    - **} else {**
        - **withholdingRate = 0.05;**
    - **cout << “$” << (withholdingRate * amtEarned) << “ will be withheld.” << endl;**
    - **}**
- **}**
- {statement, statement, statement} - compound statement/block
- **cout << “What is your name? “;**
- **string name;**
- **getline(cin, name);**
- **if (name == “”) {**
    - **cout << “You didn’t type a name!” << endl;**
- **} else {**
    - **cout << “Hello, “ << name << endl;**
- **}**
- Variables inside curly braces can only be used inside those same curly braces
- Assignment statement: variable = expression;

10/11: Discussion 2

- **#include <iostream>**
- **using namespace std;**
- **int main() {**
    - **cout << "How old are you?"**
    - **int age;**
    - **cin >> age;**
    - **if (age > 18) {**
        - **cout << "In the US, you can vote!";**
    - **} else if (age > 62) {**
        - **cout << "In the US, you can receive Social Security!";**
    - **} else**
    - **{**
        - **cout << "In the US, you can't do anything!";**
    - **}**
- **}**
    - Logic errors:
        - Order of age conditions
        - >= for ages
        - Delete else in second condition and nest the second if into the first
    - Stylistically:
        - adding << endl to the cout statements
        - standardizing the convention used for { }
- Boolean - bool, true = any non-zero value, false = 0
- Undefined Behavior: dividing by 0, using uninitialized variables, overflow

10/14: Control Flow 2

- Magic number: a number that is imposed by external requirements and is subject to change
- **int main() {**
    - **const double PAYRATE_THRESHOLD = 14.00;** // const blocks variable's reassignment, generally named with full caps, must be initialized in the same line as the declaration
    - **const double HIGH_WITHHOLDING_RATE = 0.10;**
    - **const double LOW_WITHHOLDING_RATE = 0.05;**
    - **cout << "How many hours did you work? ";**
    - **double hoursWorked;**
    - **cin >> hoursWorked;**
    - **cout << "What is your hourly rate of pay? ";**
    - **double payRate;**
    - **cin >> payRate;**
    - **double amtEarned = hoursWorked * payRate;**
    - **cout.setf(ios::fixed);**
    - **cout.precision(2);**
    - **cout << "You earned $" << (amtEarned) << endl;**
      double withholdingRate;
    - **if (payRate >= PAYRATE_THRESHOLD) {**
        - **withholdingRate = HIGH_WITHHOLDING_RATE;**
    - **} else {**
        - **withholdingRate = LOW_WITHHOLDING_RATE;**
    - **cout << "$" << (withholdingRate * amtEarned) << " will be withheld." << endl;**
    - **}**
- **}**
- **int m = 2;**
- **int n = 3;**
- …
- **n = 4 * m;** // n is now 8
- …
- **n = 2 * n;** // n is now 16
- **int a = 3;**
- **int b = a + 5;** // b is now 8
- …
- **a = 4;** // b is still 8
- **x = 3;** // can be read as "assign 3 to x" or "set x to 3" or "x gets 3"
- **double x = 4;**
- **int i = 3.8;** // i is now 3, fraction is dropped, not rounded
- **string citizenship;**
- **int age;**

- …
- **if (citizenship == “US”) {**
    - **if (age >= 18)**
        - **cout << “You can vote in U.S. elections” << endl;**
- **} else {**
    - **cout << “You are not a U.S. citizen” << endl;**
- **}** // else matches the nearest preceding if without an else, ambiguous, fix w/ curly braces
- expression && expression = “and”
- expression || expression = “or”
- **if (citizenship == “US” && age >= 18)**
    - **cout << “You can vote in U.S. elections” << endl;**
- Issue:
    - **if (a / b + c / d > 10)** // possible division by zero
        - …
    - **if (a / b + c / d > 10 && b != 0 && d != 0)** // still no good, expression will be evaluated first, division by zero still occurs
        - A && B or A || B, A is evaluated first
    - **if (b != 0 && d != 0 && a / b + c / d > 10)** // fixed
- Issue:
    - **if (citizenship == “US” || citizenship == “Canada”)**
    - **if (citizenship == “US” || == “Canada”)** // syntax error
    - **if (citizenship == “US” || “Canada”)** // wrong, will compile, always true
    - **if (age == 18 || age == 19 || age == 20)** // fine
    - **if (age == 18 || 19 || 20)** // wrong, will compile, always true, in numbers, 0 is false, anything else is true
- Issue:
    - **if (citizenship == “US” || citizenship == “Canada” && age >= 18)** // && has higher precedence than ||, therefore in A || B && C, B && C evaluates first, overridden w/ parentheses
- Issue:
    - **if (citizenship != “US” || citizenship != “Canada”)** // logic error, every string is not equal to US or Canada
    - **if (citizenship != “US” && citizenship != “Canada”)** // correct
        - Be careful when converting to opposite
        - De Morgan’s Laws:
            - not (A and B) → (not A) or (not B)
            - not (A or B) → (not A) and (not B)
    - **if (citizenship == “US” && age >= 18)**
        - **cout << “You can vote in U.S. elections” << endl;** // want opposite of this
    - **if (citizenship != “US” || age <= 18)**
        - **cout << “You cannot vote in U.S. elections” << endl;** // not right, opposite of >= is <, not <=

- not (a > b) → a <= b
- not (a >= b) → a < b
- not (a < b) → a >= b
- not (a <= b) → a > b
- not (a == b) → a != b
- not (a != b) → a == b

- Issue:
  - **if (income < 30000) {**
    - **cout << "Low";**
  - **} else {**
    - **if (income >= 30000 && income < 100000) {** // income >= 30000 unnecessary, would've been caught earlier
      - **cout << "Middle";**
    - **} else {**
      - **if (income >= 100000 && income < 500000) {** // income >= 100000 also redundant
        - **cout << "High";**
      - **} else {**
        - **cout << "Very high";**
      - **}**
  - Reformat:
    - **if (income < 30000)**
      - **cout << "Low";**
    - **else if (income < 100000)**
      - **cout << "Middle";**
    - **else if (income < 500000)**
      - **cout << "High";**
    - **else**
      - **cout << "Very High";**
        - "if ladder"

- Issue:
  - **int n = 17;**
  - **cout << "n is " << n << endl;**
  - **if (n = 0)** // doesn't compare n to 0, sets n to 0, the result is 0, therefore evaluates as false
    - **cout "n is zero" << endl;**
  - **else**
    - **cout << n is not zero; n is " << n << endl;**

10/16: Control Flow 3

- **int main() {**
    - …
    - **int choice;**
    - **cin >> choice;**
    - **if (choice == 1) {**
    - …
    - **else if (choice == 2 || choice == 4) {**
        - …
    - **} else if (choice == 3 || choice == 5) {**
        - …
    - **} else {**
        - **cout << "Choice must be 1 through 5. Goodbye." << endl;**
        - **return 1**
    - **}**
    - …
- **}**
- Returning a 0 as the exit code is conventionally a success, while a nonzero value is conventionally a failure
    - Automatically returns 0 in C++ if no exit status is specified
- **switch (choice) {**
    - **case 1:**
        - ...
        - **break;**
    - **case 2:**
    - **case 4:**
        - …
        - **break;**
    - **case 3:**
    - **case 5:**
        - …
        - **break;**
    - **default:**
        - **cout << "Choice must be 1 through 5. Goodbye" << endl;**
        - **return 1;**
- **}**
    - Switch statement will keep running through cases until end, must exit out of switch statement with break
    - Choice must be of some kind of integer type, strings won't compile
    - Must list as individual cases, cannot combine multiple numbers in one case
- **cout << "How many times do you want to be greeted? ";**
    - **int nTimes;**
    - **cin >> nTimes;**

    - **int n = 0;**
    - **while (n < nTimes) {**
        - **cout << “Hello” << endl;**
        - **n += 1;**
    - **}**
- **n = n * 2;** is the same as **n *= 2;**
- **n = n - 7;** is the same as **n -= 7;**
- **n = n + 1;** is the same as **n += 1;**
- **n++** and **++n** add 1 to n
- **n--** and **--n** subtract 1 from n
- Be careful when using starting situation and conditions together, make sure they give the result you’re intending
- do-while loop:
    - **do**
        - statement
    - **while (**condition**);**
- **}**
- for loop generally chosen when the iterator statement is simple and related to the stay-in-loop condition
- **for (int r = 0; r <= 3; r++) {**
    - **for (int c = 1; c <= 4; c++) {**
        - **cout << “*”;**
    - **}**
    - **cout << endl;**
- **}**
    - Be careful when dealing with nested loops, make sure you know what needs to happen every iteration/once only/etc.
- **string s = “Hello”;**
- **for (int k = 0; k != s.size(); k++)**
    - **cout << s[k] << endl;**
- Enter some text: *Everyone, hello!*
- The number of E’s (upper and lowercase) is 4
    - **cout << “Enter some text: “;**
    - **string t;**
    - **getline(cin, t)**
    - **int numberOfEs = 0;**
    - **for (int k = 0; k != t.size(); k++) {**
        - **if (t[k] == ‘e’ || t[k] == ‘E’) {**
            - **numberOfEs++;**
        - **}**
    - **}**
    - **cout << “The number of E’s (upper and lowercase) is “ << numberOfEs << endl;**

        - Characters use single-quotes
        - Characters left alone in and/or statements will always evaluate as true
- **string s = “Hello”;** // s is a string; “Hello” is a string literal
- **char ch = s[1];** // ch is a char, initialized to a lowercase ‘e’, ‘E’ is a character constant
- ‘\t’ = tab, ‘\n’ = newline
- characters are not strings, strings are not characters
    - **char c = ‘A’;** // ok
    - **char c = “A”;** // will not compile
    - **string s = “A”;** // ok
    - **string s = ‘A’;** // will not compile
- **#include <cctype>** // declares isdigit, isalpha, etc
    - **isdigit(**character**)** tests true for the 10 digit characters
    - **islower(**character**)** tests true for lowercase letters
    - **isupper(**character**)** tests true for uppercase letters
    - **isalpha(**character**)** tests true for any letter
    - **toupper(**character**)** gives the uppercase version of the character
        - returns character unchanged if it isn’t a lowercase character

10/18: Discussion 3

- **while (**stay-in-loop-condition**) {**
    - [code block]
- **}**
- **int x = 0;** // initializer
- **while (x < 10) {** // stay-in-loop-condition
    - [code block]
    - **x++;** // increment/decrement
- **}**
- **for (**initializer**;** stay-in-loop-condition**;** prepare-for-next-iteration**) {**
    - [code block]
- **}**
- the **break** keyword terminates the enclosing loop or switch statement
- **#include <iostream>**
- **#include <string>**
- **using namespace std;**
- **int main() {**
    - **string guess;**
    - **int tries;**
    - **for (tries = 0; tries < 3; tries++) {**
        - **cout << "Enter password: ";**
        - **getline(cin, guess);**
        - **if (guess == "secret123") {**
            - **break;**
        - **}**
    - **}**
    - **if (tries == 3) {**
        - **return 1;**
    - **} else {**
        - **return 0;**
    - **}**
- **}**
- **#include <iostream>**
- **#include <string>**
- **using namespace std;**
- **int main() {**
    - **string password;**
    - **string guess;**
    - **do {**
        - **cout << "Enter password: ";**
        - **getline(cin, guess);**
        - **if (guess == password) {**
            - **return 0;**

        - **}**
    - **} while (guess != password);**
- **}**

10/21: Functions 1

- **void greet() {**
    - **for (int k = 1; k <= 3; k++)**
        - **cout << “Hello” << endl;**
    - **}**
- **}**
- **int main() {**
    - **greet();**
    - …
    - **greet();**
    - …
    - **greet();**
- **}**
- Functions must be declared prior to use
- Declaring function before implementation → function prototype
    - Not needed for recursive functions
- No nesting functions → compilation error
- void greet (int n) ← parameter, more flexible use of a function

10/23: Functions 2

- Predicate naming convention
    - isdigit(s[k])
    - isValidPhoneNumber(phone)
    - hasMoreThanTwoPrimeFactors(n)
    - livesIn(personName, city)
- Functions have only one return type
- double& → reference to a double, another name for some already existing double
- “passing by value” - copying, no effect on original argument
- “passing by reference” - no copy, the function uses the original
- break: breaks out of the nearest enclosing loop or switch
- continue: abandons the rest of the loop’s current iteration, moves on to the next one

10/25: Discussion 4

- Chars
  - **char ch = '2';**
  - **char letter = 'a';**
  - **letter++;** // legal, increments the ASCII representation of the char
  - **cout << letter;** // prints out 'b'
  -
  - **char nine = '9';**
  - **char eight = '8';**
  - **cout << nine - eight;** // evaluates as ASCII value of 9 minus the value of 8, prints '1'
- Strings
  - Sequence of chars
  - **string str = "w$.]";**
  - **char ch = str[4];** // ch = ']'
  - **string str1 = str[3];** // doesn't work, no built-in conversion from string to char
  - **string str2 = "a";** // allowed
  - **string str3 = 'a';** // not allowed
- String Functions
  - size():
    - **string str = "Hello";**
    - **int size = str.size();** // size = 5
  - isdigit():
    - returns non-zero int if digit (0-9)
  - isalpha():
    - returns non-zero int if letter
  - islower():
    - returns non-zero int if lowercase letter
  - isupper():
    - returns non-zero int if uppercase letter
  - toupper():
    - returns uppercase letter if char is lowercase
  - tolower():
    - returns lowercase letter if char is uppercase
- Functions
  - Reduce repeated code
  - Organized/readable
  - Easier to debug/test
  - Cannot nest functions in C++
  - Function signature:
    - **returnType functionName(paramType1 paramName1, paramType2 paramName2) {**
    - ...

        - **}**
    - Call a function:
        - **functionName(varName1, varName2);**
- Passing by value
    - Through separate variables with equal values
    - Changing the new variable doesn't modify the original one
- Passing by reference

11/1: Arrays

- Array length must be known in compile time
- **cout << "Enter the scores (negative when done):" << endl;**
- **const int MAXSCORES = 10000;**
- **int scores[MAXSCORES];** // declare array with enough spaces so that any run of the program will be fine
- **int total = 0;**
- **int nScores = 0;** // have a variable to tell you how many interesting elements are in the array, tells the position of the next available spot
- **for (;;) {**
    - **int s;**
    - **cin >> s;**
    - **if (s < 0) {**
        - **break;**
    - **}**
    - **if (nScores == MAXSCORES) {**
        - **cout << "I can handle only " << MAXSCORES << " scores!" << endl;**
        - **cout << "Continuing with just the first " << MAXSCORES << "values." << endl;**
        - **break;**
    - **scores[nScores] = s;**
    - **total += s;**
    - **nScores++;**
- **}**
- **if (nScores == 0) {**
    - **cout << "There were no scores, so no statistics" << endl;**
- **} else {**
    - **double mean = static_cast<double>(total) / nScores;**
    - **cout << "The average of the scores is " << static_cast<double>(total) / nScores << endl;**
    - **double sumOfSquares = 0;**
    - **for (int k = 0; k < nScores; k++) {** // make sure the condition is correct
        - **double diff = scores[k] - mean;** // for all times accessing an array, make sure the subscript is always within bounds (can't be negative or above/equal to the maximum elements in the array
        - **sumOfSquares += diff * diff;**
    - **}**
    - **cout << "The std. deviation is " << sqrt(sumOfSquares / nScores) << endl;** // when using /, make sure there will never be a divide by zero, when using functions make sure the parameters will always be correct
- **}**

- **double computeMean(const int a[], int n) {** // arrays are never passed by value, function only knows where the array starts, impossible to check if n is too big, const promises that the elements in a are not modified in the function
    - **if (n<= 0) {**
        - **return 0;**
    - **int total = 0;**
    - **for (int k = 0; k < n; k++) {**
        - **total += a[k];**
    - **return static_cast<double>(total) / n**
- **int main() {**
    - …
    - **const int MAXSCORES = 10000;**
    - **int scores[MAXSCORES];**
    - **int nScores = 0;**
    - … // fill up the score array
    - **double m = computeMean(scores, nScores);**
    - …
    - **int stuff[100];**
    - …
    - **double m2 = computeMean(stuff, 100);**
- **}**
- When passing an array into a function, it could be just looking at the array or modifying it
- **void setAll(int b[], int n, int value) {**
    - **for (int k = 0; k < n; k++) {**
        - **b[k] = value;**
    - **}**
- **}**
- **const int daysInMonth[12] = {31, 28, 31, 30, …, 31};**
- **cout << computeMean(daysInMonth, 12) << endl;** // allowed
- **cout << setAll(daysInMonth, 12, 30);** // compilation error, cannot pass const int array to function that may modify the array
- Calling a function that may modify an array within a function that promises to not modify the array ends in a compilation error
- **#include <string>**
- **using namespace std;**
- **string t = “Hello”;**
- **string s;** // empty string
- **for (int k = 0; k != t.size(); k++) {**
    - **cout << t[k] << endl;**
- **}**
- **cout << t;**
- **getline(cin, s);**
- **s = t;** // sets s to Hello, automatically adjusts the size of a string

- **s += “!!!”** // sets s to Hello!!!
- **if (t < s)** // When matching, shorter strings are less than longer ones, otherwise first differing character decides w/ letter coming first as less
- ‘0’ < ‘1’, ‘1’ < ‘2’ … ‘8’ < ‘9’, contiguous values
- ‘a’ < ‘b’, ‘b’ < ‘c’ … ‘y’ < ‘z’ // not guaranteed to be contiguous
- ‘A’ < ‘B’, ‘B’ < ‘C’ … ‘Y’ < ‘Z’ // no guarantees if uppercases are less than lowercases, ASCII has all uppercases less than lowercases
- ‘ ‘ < ‘a’, ‘ ‘ < ‘A’, ‘ ‘ < ‘0’

11/4: C Strings

- '\0' - zero byte - marks the end of a string
- **char t[10] = { 'H', 'e', 'l', 'l', 'o', '\0' };** // don't have to initialize every value, uninitialized values will be given a neutral value
    - Shorthand: **char t[10] = "Hello";** —> same as above, zero byte implied
    - Key difference: you have to think about the length of the string + the zero byte
- **char s[100];** // will be an array of 100 uninitialized characters, not the empty string
    - **char s[100] = "";** or **char s[100] = { '\0' };** or **char s[100] = { };** initialize it as an empty string
- **for (int k = 0; t[k] != '\0' ; k++) {**
    - **cout << t[k] << endl;**
- **cout << t;** // outputs each character up to, but not including, the zero byte
- **cin.getline(s, 100);** // name of the char array and a limit for how many characters the array can store, including the zero byte
- **s = t** // doesn't work, cannot assign arrays to each other in C or C++
- **#include <cstring>** - includes functions to use with C strings
    - **strlen(t);** - returns the length of C string t, same as **t.size()** for C++ strings
        - t has to be an array of characters for strlen()
    - **strcpy(s, t);** - s is the destination, t is the source, copies the string
        - Undefined behavior if the destination is not big enough
        - **strcpy(t, "Goodbye");** - the source string can be a string literal
    - **strcat(s, "!!!");** - s is appended with "!!!"
        - Second argument has to be a C string (zero byte at the end)
        - Destination has to be big enough to hold the result
- **#define _CRT_SECURE_NO_WARNINGS** - for a Microsoft compiler to stop warnings when using <cstring>
- Cannot compare C strings with (s == t), (s < t), etc.
    - Will compile, will not perform intended function
    - **if (strcmp(t, s) < 0)** same as **if (t < s)** for C++ strings
        - **strcmp(a, b)** is negative if a comes before b, 0 is a equals b, and positive if a comes after b
        - Mistake: **if (strcmp(a, b))** —> strcmp() returns an int not a bool, yields the opposite result (returns false if equal)
- **const int NWEEKS = 5;**
- **const int NDAYS = 7;**
- **int attendance[NWEEKS][NDAYS];** // declares a 2D array with 5 rows and 7 columns
- ...
- **cout << attendance[2][5];**
- **for (int w = 0; w < NWEEKS; w++) {**
    - **int t = 0;**
    - **for (int d = 0; d < NDAYS; d++) {**
        - **t += attendance[w][d];**
    - **cout << "The total for week " << w << " is " << t << endl;**

- **}**
- **const string dayNames[NDAYS] = { “Monday”, “Tuesday”, …, “Sunday” };**
- **int grandTotal = 0;**
- **for (int d = 4; d < NDAYS; d++) {**
    - **int t = 0;**
    - **for (int w = 0; w < NWEEKS; w++) {**
        - **t += attendance[w][d];**
    - **cout << “The total for “ << dayNames[d] << “ is “ << t << endl;**
    - **grandTotal += t;**
- **}**
- **cout << “Over the course of “ << NWEEKS << “ weeks, weekend attendance was” << grandTotal << endl;**

11/6: 2D Arrays and Pointers

- **double meanForADay(const int a[][NDAYS], int nRows, int dayNumber) {** // have to write the number of columns, not the number of rows
    - **if (nRows <= 0) {**
        - **return 0;**
    - **}**
    - **int total = 0;**
    - **for (int r = 0; r < nRows; r++) {**
        - **total += attendance[r][dayNumber];**
    - **}**
    - **return static_cast<double>(total) / nRows;**
- **int main() {**
    - **int attendance[NWEEKS][NDAYS];**
    - …
    - **double m = computeMean(attendance[2], NDAYS)** // 2D arrays are treated like an array of 1D arrays, not the same with columns of the 2D array
    - …
    - **double m2 = meanForADay(attendance, NWEEKS, 4);**
    -
- }

- **int multiplexAttendance[NWEEKS][NDAYS][16];**
- **void f (int b[][NDAYS][16],** … **);** // passing a 3D array
- **int chainAttendance[NWEEKS][NDAYS][10][16];**
- **string cppstrings[5] = { “aaa”, “bbb”, “ccccc”, “”, “eeee” }**

- **const int MAXWORDLENGTH = 6;** // account for the zero byte later
- **int findFirstOfLength(char a[][MAXWORDLENGTH + 1], int nRows, int targetLength) {**
    - **for (int r = 0; r < nRows; r++) {**
        - **if (strlen(a[r]) == targetLength) {**
            - **return r;**
        - **}**
    - **}**
    - **return -1;**
- **}**
- **int main() {**
    - **const int MAXPETS = 5;**
    - **char pets[MAXPETS][MAXWORDLENGTH + 1] = { “cat”, “mouse”, “eel”, “ferret”, “horse” }**
    - **cout << findFirstOfLength(pets, MAXPETS, 5)**
- **}**

- **int findFirstOfLength(string a[], int nRows, int targetLength) {** // C++ version
    - **for (int r = 0; r < nRows; r++) {**
        - **if (a[r].size() == targetLength) {**
            - **return r;**
        - **}**
    - **}**
    - **return -1;**
- **}**

- Pointers:
    - Another way to implement passing by reference
    - Traverse arrays
    - Manipulate dynamic storage
    - Represent relationships in data structures
- **#include <cmath>**
- **using namespace std;**
- **void polarToCartesian(double rho, double theta, double* xx, double* yy);**
- **int main() {**
    - **double r;**
    - **double angle;**
    - … get values for r and angle …
    - **double x;**
    - **double y;**
    - **polarToCartesian(r, angle, &x, &y);**
    - …
- **}**
- **void polarToCartesian(double rho, double theta, double* xx, double* yy) {**
    - ***xx = rho * cos(theta);**
    - ***yy = rho * sin(theta);**
- **}**
- double* means pointer-to-double or address-of-some-double
- &x means “generate a pointer to x” or “address of x”
- *p means “the object that p points to” or “follow the pointer p”

- **double a = 3.2;**
- **double b = 5.1;**
- **double* p = &a;**
- **double* q = b;** // wrong, type mismatch
- **double c = a;**
- **double d = p;** // wrong, type mismatch
- **double d = *p;**
- **p = b;** // wrong, type mismatch
- **p = &b;** // p now holds a pointer to b

- ***p = b;** // the object that p points to (a) now has the value of b
- ***p += 4;**
- **int k = 7;**
- **p = &k;** // p can only hold pointers to doubles, cannot convert to a pointer to an int
- **int* z = &k;**

11/13: Pointers

- **k = 7; a = 3.2; b = 9.1; c = 3.2; d = 3.2; int* z = &k, double* p = &b;**
- **cout << (k * b);** // writes 63.7
- **cout << (k * p);** // won't compile
- **cout << (k * *p);** // writes 63.7
- **cout << (*z * *p);** // writes 63.7
- **double* q;**
- ***q = 4;** // undefined behavior, may crash right away, may write to a random place in memory, may write to a used address in your program
- **double* r = &a;**
- ***r = b** // a gets b
- **if (p == r)** // false, p points to b, r points to a
- **q = p;**
- **if (p == q)** // true, pointers point to the same address
- **if (*p == *r)** // true, values of doubles that p and r point to are equal

- **const int MAXSIZE = 5;**
- **double da[MAXSIZE];**
- **int k;**
- **double* dp;**
- **for (k = 0; k < MAXSIZE; k++) {**
    - **da[k] = 3.6;**
- **}**
- &a[i] + j = &a[i +j]
- &a[i] < &a[j] if i < j, depends on the subscript within the array
    - Ok to do, as long as the pointers point to elements within the array or one past the end of the array
- **for (dp = da; dp < da + MAXSIZE; dp++) {** // pointer version
    - ***dp = 3.6;**
- **}**
- When using the name of the array, it is treated as a pointer to element 0 of that array (see loop conditions above)

- **string b[6] = { "rat", "pig", "goat", "zebra", "pig", "rat" };**
- **int m = lookup(a, 6, "zebra");** // 3
    - int lookup(const int a[], int n, string s) → as a parameter, the first parameter calls for a pointer to array a
- **int k = lookup(&b[2], 3, "zebra");** // 1
- p[i] = *(p + i)

11/20: Classes

- &a[i] - &a[j] = i - j
- nullptr gives the null pointer value, used for checking purposes
    - Not an uninitialized variable
- **struct Employee {**
    - **string name;**
    - **double salary;**
    - **int age;**
- **};** ←---------- don't forget the semicolon
- When declaring a data member as a string and initializing an array, the string values will be initialized as empty strings, while int/double are garbage
- The . operator takes an object of some structure type on the left and the name of a member of that type on the left

- **struct Target {**
    - **int pos;**
    - **string history;**
    - // Invariant:
    - // history consists of only Rs and Ls
    - // pos == number of Rs in history minus number of Ls in history
- **};**
- **int main() {**
    - **Target t;**
    - **init(t);**
    - **moveRight(t);**

11/25: Classes

- **struct Target {**
  - **public:** // public members can be used anywhere in the program
  - **void init();** // member functions
  - **bool move(char dir);**
  - **void replayHistory() const;**
  - **int position() const;** // declares position as a const member function
  - **private:** // private members can only be used in the implementations of member functions of the same type
  - **int pos;** // data members
  - **string history;**
    - // Invariant:
    - // history consists of only Rs and Ls
- **};**
- **void Target::init() {**
  - **this->history = “”;** // this-> not necessary
  - **this->pos = 0;**
- **}**
- **bool Target::move() {**
  - **switch (dir) {**
    - **case ‘R’:**
    - **case ‘r’:**
      - **this->pos++;**
      - **break;**
    - **case ‘L’:**
    - **case ‘l’:**
      - **this->pos--;**
      - **break;**
    - **default:**
      - **return false;**
  - **}**
  - **this->history += toupper(dir);**
  - **return true;**
- **}**
- **void Target::replayHistory() const {**
  - **for (int k = 0; k != this.history.size(); k++) {**
    - **cout << this.history[k] << endl;**
  - **}**
- **}**
- **void repeatMove(Target& x, int nTimes, char dir) {**
  - **for (int k = 0; k < nTimes; k++) {**
    - **x.move(dir);**
  - **}**

- **}**
- **int Target::position() const {**
    - **return this->pos;**
- **}**
- **void report (const Target& x) {** // calling x.position() with const creates a compilation error
    - **cout << “There’s a target at position “ << x.position() << endl;**
- **}**
- **int main() {**
    - **Target t;**
    - **t.init();** // passes a pointer to the object you are calling the function on
    - **t.move(‘R’);**
    - **repeatMove(t, 3, ‘L’);**
    - **report(t);**
    - **Target t2;**
    - **t2.init();**
    - **t2.move(‘L’);**
    - **t2.replayHistory();**
- **}**
- Data members can be made private so that implementation can be changed without changing utility
- struct and class are the same things, almost interchangeable
    - structs default to public
    - classes default to private

12/2: Classes

- **class Pet {**
  - **public:**
  - **Pet(string nm, int initialHealth);**
  - **string name() const;**
  - **private:**
  - **string m_name;**
  - **int m_health;**
- **};**
- **Pet::Pet(string nm, int initialHealth) {**
  - **m_name = nm;**
  - **m_health = initialHealth;**
- **}**
- **string Pet::name() const {**
  - **return m_name;**
- **}**
- **void f(string s) {**
  - **Pet p(s, 10);**
- **}**
- **int main() {**
  - **while(...) {**
    - **string t;**
    - **getline(cin, t);**
    - **f(t);**
  - **}**
- **}**

- **class Target {**
  - **public:**
  - **Target();**
  - **void replayHistory() const;**
  - **private:**
  - **string m_history;**
- **};**
- **Target::Target() {**
  - **m_history = “”;**
- **}**
- **void Target::replayHistory() const {** // don’t make k a data member, it doesn’t need to be remembered between calls of member functions
  - **for (int k = 0; k != m_history.size(); k++) {**
    - **cout << m_history[k] << endl;**
  - **}**

- **}**

- **class Tank {**
    - **public:**
    - **Tank(double gallons);**
- **}**
- **Tank tk(1000);**

- When calling a constructor on a class with no parameters, you call it without any parentheses
- Dynamically allocated objects exist until you explicitly delete them

- **class Employee {**
    - **public:**
    - **Employee(string nm, double sal, int ag, Company* cp);**
    - **void receiveBonus() const;**
    - **private:**
    - **string m_name;**
    - **double m_salary;**
    - **int m_age;**
    - **Company* m_company;**
- **};**
- **class Company {**
    - **public:**
    - **Company();**
    - **void hire(string nm, double sal, int ag);**
    - **void setBonusRate(double pct);**
    - **void giveBonuses() const;**
    - **double bonusRate() const;**
    - **private:**
    - **Employee* m_employees[100];**
    - **int m_nEmployees;**
- **};**
- **Company::Company() {**
    - **m_nEmployees = 0;**
- **}**
- **void Company::hire(string nm, double sal, int ag) {**
    - **m_nEmployees[m_nEmployees] = new Employee(nm, sal, ag, this);**
    - **m_nEmployees++;**
- **}**
- **Employee::Employee(string nm, double sal, int ag, Company* cp) {**
    - **m_name = nm;**
    - **m_salary - sal;**

        - **m_age = ag;**
        - **m_company = cp;**
- **}**
- **void Employee::receiveBonus() {**
    - **cout << “Pay to the order of “ << m_name << “ the amount $ “ << (m_salary * m_company->bonusRate());**
- **void Company::setBonusRate(double pct) {**
    - **m_bonusRate = pct;**
- **}**
- **void Company::giveBonuses() {**
    - **for (int k = 0; k != m_nEmployees; k++) {**
        - **m_employees[k]->receiveBonus();**
    - **}**
- **}**
- **double Company::bonusRate() const {**
    - **return m_bonusRate;**
- **}**