COP 3363

SPRING 2020

RECITATION 6

FUNCTIONS

- Two primary roles: abstraction & reusability
- C++ format: <return type> <function name>(parameter list)
- Utilize all of the concepts (and more) which have been introduced thus far
  - variables, i/o, selection, repetition, basic math operations
- (Typically) accepts some input and (typically) returns some output

ATMS

- Many different real world activities follow a functional pattern
- Using an ATM
  - input: a debit card, a pin number, a $ amount
  - output: currency which is equivalent to that $ amount
  - C++ format: dollars atm_withdrawl(card, pin, amount)

ABSTRACTION

- Makes code easier to read, distribute, and reuse
- A core concept behind function usage is that a user (programmer) can trust the result of a function's execution without knowing the steps of the execution
    - ex. sin(x), cos(x), setprecision(2)
- Fewer and cleaner lines of code in your main routine make your source code easier to comprehend
    - Imagine you have a 10 line menu which has to print 5 times in your program. Is it more sensible to dedicate 50 or 5 lines to that in your main routine?

REUSABILITY

- Programs typically involve lots of repetition. How to make that easier?
  - Loops and functions
    - Loops make repetition easier at a particular point in your program
    - Functions make repetition easier when it takes place in different parts of your program.

RETURN VS PRINTING

- Returning data from a function and printing to the screen (cout) from within a function are not the same thing
  - Using cout in a function simply prints data to the screen just like in main()
  - Returning data from a function does not display on screen
    - ex. double x; x=sin(3); cout << x;
    - x stores the value of sin(3) after the second statement, but you don't see it on screen until the third
    - atm example: returning - getting the physical currency, printing - seeing the withdraw amt print on the screen

DIFF

- Format: diff [flags] <original file> <newfile>
- Compares 2 files or directories and prints lines where there is a difference
- Useful flags
  - b: Treats groups of spaces as one
  - i: Ignores case
  - r: Includes directories in comparison
  - w: Ignores all spaces and tabs

GREP

- Searches files for a particular pattern. The pattern can be a word, a string enclosed in single quotes, or a regular expression.
  - grep int *.c (find all occurences of the pattern 'int' in all files with a .c extenstion)
  - grep 'm.*n' myfile (the . matches a single character, the .* matches any number of characters; this finds anything starting with an m and ending with an n)
- Useful flags:
  - i: ignore case
  - n: display the line numbers
  - l: display only names of files and not actual lines

TAR

- Create and extract file archives
- tar [flags] <archive name> <files>
- Useful flags:
  - c: insert files into a tar file
  - v: output the name of each file as it is inserted into or
  - f: use the name of the tar file that is specified
  - x: extract the files from a tar file

WILDCARDS * ? [ ]

- ? matches any single character in a filename
  - b?t will match bit, bot, bat. It will not match bt or boot
- * matches any number of characters in a filename
  - con* will match con, condor, constant.exe
  - *.c will match all files that end in .c
- [] will match any one of the characters in the brackets.
  - A hyphen “-” can be used to match any of a range of consecutive characters.
  - [bhr]at will match bat, hat and rat
  - chap[5-8].c will match chap5.c, chap6.c, chap7.c and chap8.c