Libraries

Sunday, November 1, 2015

9:03 PM

* #include <stdio.h>
  + "Standard Input/Output"
  + Includes all functions that control standard input and output
    - scanf
      * scanf("<formatcode>", &variable1, &variable2);
        + Input function.
        + Reads input from the user.

User must press "Enter" for their input to go through and for the rest of the program to continue.

* + - * + No angle brackets are needed. Only format codes to be stored into variables can be used. Ampersand is a MUST, however.

Likewise with printf, order matters.

* + - * + e.g.

printf("Please enter your age.\n");

scanf("%d", &userAge);

OR

printf("Enter today's date: ");

scanf("%d%d%d", &month, &day, &year);

* In regards to strings, specifically:
  + scanf("%s%s%s", variable1, variable2, variable3)
    - scanf("%s", word);
    - Note that an ampersand is unneeded here because an array variable is really a memory address in C already.

* printf
  + printf("text", variable1, variable2);
    - Output function.
    - The first parameter is a string, and the other parameters are optional parameters whose types correspond to the variables designated to print out by the printf function.
    - Printf returns an integer representing the number of bytes printed, but typically speaking, it's not a value we need to know. If a function's return type is void, or if we do not need to use the information the function returns to us, we typically call the function on a line by itself, as printf is usually called.
    - Prints a string literal to the screen.
      * A *string literal* is a string of characters and is denoted by the characters inside of double quotes.
    - "text" *also* includes format codes (see below) and escape characters.
      * variables to substitute for format codes are included after quotation marks and a comma
        + Each variable is separated by a comma, and each variable will be used in conjunction and order with the format codes, i.e., the first format code will be substituted by the first variable, and so forth.

* **Important Escape Sequences**
  + \n
    - new line
  + \\
    - prints **one** backslash
  + \"
    - prints quotation marks
  + e.g. printf("Hi, my name is Kevin and I am %d years old!\n", age);
  + See "Frivolous Codes" for more escape sequences.

* **Formatting Output Spacing**
  + printf("%c%3c%3c", 'A', 'R', 'G');
    - This will print out:
      * A R G
    - To further specify, printf allocated 3 spaces after 'A' and since there was only one letter, namely 'R', to fill up those 3 spaces, that letter 'R' took up the rightmost space, leaving two empty spaces to the left.
      * In total, there are still three spaces, but there are two empty spaces and one "filled" space.
  + %10.3lf
    - Allocates 9 spaces before the decimal point (to the left) and 3 spaces after the decimal.
      * The 10th space includes the decimal point.
    - The last digit to display after the decimal is rounded.
      * e.g. 3.2355 would be displayed as 3.236
      * e.g. 3.2354 would be displayed as 3.235
* malloc

* #include <stdlib.h>
  + "Standard Library"
  + Includes:
    - calloc(n, sizeof(int))
      * n is the number of elements
      * sizeof is a keyword, which takes a data type as a parameter
    - malloc(sizeof(int) \* n)
      * as above
    - ptr = realloc(ptr, 2\*n\*sizeof(int))
      * doubles the current allocation for ptr by multiplying by 2
    - Rand
      * rand( );
        + This function takes in 0 parameters and returns a single integer.
        + The integer returned is a random integer between 0 and 32767 (215 - 1).
        + int x = 1 + rand( ) % 100;

Picks a random integer in between 1 and 100.

More complexly, what this does is that rand will choose a random integer. Any chosen non-negative integer mod 100 will return a value between 0 and 99 inclusive. Adding 1 to this yields an integer between 1 and 100 inclusive.

* + - * + However, the code above will yield the same random number every time since there is no such thing as truly random numbers.
        + The random number generator in C is known as a psuedorandom number generator that uses a formula to "calculate" the next random number. But, in order to use that formula, it must have a starting value, known as a seed. Unfortunately, if no seed is given, as is the case here, the same seed is used every time.

* srand
  + srand( );
    - This void function seeds the random number generator.
    - srand(time(0));
      * Provides the elapsed time since January 1, 1970 for seeding, meaning that it takes the elapsed time, and it plugs this value into the formula for calculating a random number.
      * This line is usually included near the beginning of function main for any program that utilizes the rand function to ensure different executions of the program produce different psuedorandom numbers.
    - See page 157 of *Programming Knights: An Introduction to Computer Programming in Python and C* for other methods of seeding the random number generator.

* #include <math.h>
  + "Math"
  + Includes:
    - abs
      * abs( );
        + This function takes in a variable or number as its parameter. This function then returns the absolute value.
        + variable = abs(-14);

This will return 14. Use printf to see the result.

The left-hand side of an assignment statement MUST BE a variable.

* + - * int abs(int x);
        + This is a function prototype.

i.e., this is the formal definition of the function. This raw form cannot be used, however.

It should be noted that since this function has already been predefined, there is no need to prototype the function.

* + - * + Returns the absolute value of x.

* pow
  + pow(<parameter 1>, <parameter 2>);
    - This function takes in a variable or number for either of its parameters. This function will will raise the first parameter to the power of the second parameter.
    - variable = pow(tuna, 2);
      * According to Bucky Roberts, tuna is 20. Therefore, 202 = 400.
      * The left-hand side of an assignment statement MUST BE a variable.
  + double pow(double x, double y);
    - This is a function prototype.
      * i.e., this is the formal definition of the function. This raw form cannot be used, however.
      * It should be noted that since this function has already been predefined, there is no need to prototype the function.
    - Returns x raised to the power y.

* sqrt
  + sqrt( );
    - This function takes in a variable, number, or expression as its parameter. This function then returns the square root.
    - variable = sqrt(17);
      * This will return a value approximate to 4.123106.
      * The left-hand side of an assignment statement MUST BE a variable.
  + double sqrt(double x);
    - This is a function prototype.
      * i.e., this is the formal definition of the function. This raw form cannot be used, however.
      * It should be noted that since this function has already been predefined, there is no need to prototype the function.
    - Returns the square root of x.

* #include <string.h>
  + "String"
  + Includes:
    - strcpy
      * <http://stackoverflow.com/questions/1258550/why-should-you-use-strncpy-instead-of-strcpy>
      * strcpy(<string1>, <string2>);
        + This function takes in two strings as its parameters. The function then copies the contents of <string2> into <string1>.
        + char word[20];

strcpy(word, "hello");

* This function is best used for sorting and swapping.
  + When sorting and swapping. consider temporary variables to store data.
* int strcpy(char\* s1, const char\* s2);
  + This is a function prototype.
    - i.e., this is the formal definition of the function. This raw form cannot be used, however.
    - It should be noted that since this function has already been predefined, there is no need to prototype the function.
    - Note that since array names are nothing but memory addresses, we can simply type in the array name, as indicated above.
  + The contents of s2 are copied into s1.
  + It's important to remember that arrays are always passed by reference, so to speak, so the function itself doesn't have separate copies of the two strings passed to it. Rather, s1 and s2 are pointing to the appropriate memory addresses of the original strings as the function runs.
  + Due to the keyword const in front of the second parameter, the function strcpy can NOT make any changes to the contents of s2.
  + The function actually returns an integer, which is the memory address of the first string passed to the function.

* strlen
  + strlen(<string\_array>);
    - This function takes in a string as its parameter. The function then determines the number of characters in that string, EXCLUDING the null character.
      * This function is most commonly used in for loops.
      * This function can also be used with on a string to determine how many characters it has before '\0' as such:
        + int numCharacters = strlen("Hello");
    - variable = strlen(word);
      * Finds the number of characters stored by the string array *word*.
  + int strlen(const char\* s);
    - This is a function prototype.
      * i.e., this is the formal definition of the function. This raw form cannot be used, however.
      * It should be noted that since this function has already been predefined, there is no need to prototype the function.
    - This function returns the number of characters in s before '\0' is encountered.

* strcmp
  + strcmp(<string1>, <string2>);
    - This function takes in two strings for its parameters. The function determines which string comes first lexographically. A negative integer will be returned if the first string precedes the second string. If the two strings are identical, 0 is returned. Otherwise, a positive integer is returned.
      * The term lexographical is very similar to alphabetical, but is different in that comparisons between strings are based on the ascii values of the characters instead of the letters they represent.
        + Thus, in a lexigographical comparison, the string "apple" comes AFTER the string "Zoo" because the ascii value for the character 'Z' is 90. If two strings are either all lowercase or all uppercase, then the lexicographical comparison is the same as the alphabetical comparison, because the ascii values (65-90) of all the uppercase letters are based on alphabetical order and the ascii values of all the lowercase letters (97-122) are also based on alphabetical order.

In other words, capital letters come before lowercase letters.

Additional note: Lower ascii values come first.

* strcmp(word1, word2);
  + Compares string variables 'word1' and 'word2'.
  + Note that the shorter (but initially equivalent) string will be designated as come first, in lexographical order.
  + Also, note that most non-letter characters (such as '!'), have a lower ascii value than letters. Therefore, they will have precedence over letters in lexographical comparisons.
* int strcmp(const char\* s1, const char\* s2);
  + This is a function prototype.
    - i.e., this is the formal definition of the function. This raw form cannot be used, however.
    - It should be noted that since this function has already been predefined, there is no need to prototype the function.
  + If s1 precedes s2 lexicographically, a negative integer is returned. If the two are identical, 0 is returned. Otherwise, a positive integer is returned.

* strcat
  + strcat(<string1>, <string2>);
    - This function takes in two strings as its parameters and concatenates both strings together in order from left to right.
      * The first string has to be a variable, but the second string does not have to be.
    - strcat(word1, word2);
      * Note that no variable assignment is needed
  + char\* strcat(char \*s1, const char\* s2);
    - This is a function prototype.
      * i.e., this is the formal definition of the function. This raw form cannot be used, however.
      * It should be noted that since this function has already been predefined, there is no need to prototype the function.

* #include <time.h>
  + "Time"
  + Includes:
    - time
      * time( );
        + This function takes in an integer parameter.
        + When this function is given 0 as its parameter, then it returns the number of seconds that have elapsed since January 1, 1970.
        + printf("time(0) currently = %d\n", (0));

Produces the following output on the afternoon of February 25, 2010:

time(0) currently = 1267133747

* + - * + This function is especially useful when conjoined with the *rand* function from stdlib.

**Key**

* Section Heading
* General comment about the heading
* Further explanation about the "General comment"

* Snippets of code (examples)
* General formatting of code
* Explanation of snippets of code
* Additional remark

* Miscellaneous