Day 10 Working with Characters and Strings

if char variable used somewhere a character expected, implemented as character. if used somewhere number expected, interpreted as a number

char a,b,c char variales declared

char code = 'x' variable code assigned value of x

#define EX 10. char code = EX.

const char A = 'Z'

'a' creates literal character constant

%c to print character for printing

%d of character will return ASCII code

want to print extended ASCII characters must use unsigned char

string- sequence of characters ending with null character

so when storing in an array ensure you add array[n+1] for the null character

char *message = "This is a string"; == char message[] = "This is a string";

if do not specify number of subscripts, compiler calculates for you: char string[] = "Alabama";

since end of string marked just need something that POINTS at the beginning

using array's name to access strings is method C library expects

using array to store a string really only for allocating space

*array = array[]

malloc()- if program has dynamic memory needs. pass it the amount of memory needed. malloc() will find memory of required size and return address of first byte. return type to pointer type void- compatible all data types. **returns address of first block at return type**

char *str; str = (char *) malloc(size) ex: (int *) malloc(50 * sizeof(int)); if you just put malloc(50) it would only allocate 50 bytes

ptr = malloc(1)- allocates 1 byte of memory and assigns address of the 1st of 1 bytes to variable ptr. This byte of memory has no name- only pointer can reference. To assign a variable would need: *ptr = x;

Ex: char *ptr; ptr = malloc(100);

if not enough memory is available, malloc returns null (0)

if you want to step through the contents of a string with ptr pointing at beginning, need to keep ptr pointed at beginning, so add another variable: p = ptr; for(count = 65 ; count < 91 ; coun++) *p++ = count;

free() towards end of program in order to not hog memory space

in above example notice incremented pointer along with count so pointer moves along

puts() really only takes pointers to the string to be displayed. literal string technically a pointer to itself (weird)

%s to display a whole string. but must refer to a pointer to the string

gets() and scanf() can read from keyboard but need place to put info. create with array declaration or malloc()

gets()- will read all characters entered until newline (Enter) is pressed. will discard newline and add null character, ending string. then returns pointer to the string. gets a string from standard input device

if string has length of 0, null stored in first position

ex: while(*(ptr = gets(Input))) != NULL - will return value stored at pointed to address

scanf() also passed a pointer to strings location. beginning is first non whitespace character. runs up to next non whitespace character (not including) if use %s. if use %ns where n is an integer, accepts next n characters or up to next whitespace character whichever comes first

scanf() can be passed a pointer, array name or &variable

function prototype to return a pointer: char * function(char *, char *); ex6 requires you to find length of 2 strings then point at the longer string. creates pointers to strings, then uses strlen() to get lengths in function like exhibited here, then returns the longer with an if condition which is assigned to a pointer