**Char Arrays & Char Pointers**

// \*\*\*\*\* CHAR ARRAYS \*\*\*\*\*

// Character Arrays vs Integer Arrays

// Instead of outputting an address of the char array (like it would for ints),

// a char array will output the data.

// a Char array needs an additional element over and above the number of chars shown in the initialization.

// -- because a char array needs one index at the end of the array for a null character.

#include <iostream>

using namespace *std*;

int main()

{

// leave one additional index for the null char

char name[7] = { 'D', 'u', 'n', 'c', 'a', 'n' };

*cout* << "Your name is " << name << "." << *endl*;

// error - name = "Duncan";

// assign a string literal to initialize a char array

char name2[7] = "Duncan";

*cout* << "Your name is with a string literal passed into the char array is: " << name2 << "." << *endl*;

// String literal to a Char Pointer - makes an array but referred to as a pointer

// When c++ sees that a pointer is assigned to a string literal, it reserves a block of memory for each individual character and treats the char pointer like an array

// Since strings are read in as char arrays by compiler, and an array is just a pointer that points to the first object of that array

// instead of spitting out a memory address of the pointer, the compiler grabs the first chars value and spits out everything in array until Null is reached

char \* name3 = "Mario"; // name -> Mario\_

// Re-assigning value of char pointer only appends the value of name3, but doesn't remove first name. Allocates more memory.

// However, the pointer now points to the address of where the 'D' character resides in memory. So, it appears that Mario was removed when it wasn't.

name3 = "Duncan"; // name ->Mario\_Duncan

*cout* << "Your name is with a string literal passed in to a pointer is: " << name3 << "." << *endl*;

return 0;

}