**Lecture 21 - Chapter 8: C Characters and Strings – Wed Nov 1 or Thurs Nov 2**

**Announcements**

Reading:

* Chapter 8

Assignments:

* Due: Assignment #8
* Assign: Assignment #9 - due on **Nov 8** (MW class) or **Nov 9** (TR class) **(no late assignments accepted)**

**Today’s Goals**

1. Assignment #7 Feedback
2. Fundamentals of Strings and Characters
3. Character-Handling Library
4. String-Conversion Functions
5. Standard Input/Output Library Functions
6. String Manipulation Functions of the String-Handling Library

**Assignment #7 Feedback**

Needed to declare the following:

// Declare primitive types

**int** intValue = 1234;

**double** doubleValue = 367.7;

**char** charValue = 'A';

**int** intArray[5] = {8, 1, 7, 2, 4};

// Declare pointer types

**int** \*intValuePtr = &intValue;

**double** \*doubleValuePtr = &doubleValue;

**char** \*charValuePtr = &charValue;

// Declare pointers to pointers

**int** \*\*pointerToIntValuePtr = &intValuePtr;

// Declare pointer to integer array – first set to NULL and print, then point to array

**int** \*intArrayPtr = NULL;

intArrayPtr = intArray;

Print information for the integer array variable and each element in the array

int intArray 0028FF18 0028FF18 20 array name address 1st item

int intArray[0] 0028FF18 8 4

int intArray[1] 0028FF1C 1 4

int intArray[2] 0028FF20 7 4

int intArray[3] 0028FF24 2 4

int intArray[4] 0028FF28 4 4

// Print integer array (I removed spacing and comment to fit on line)

**printf** ("int intArray %p %p %d\n", &intArray, intArray, **sizeof** (intArray));

**printf** ("int intArray[0] %p %d %d\n", &intArray[0], intArray[0], **sizeof** (intArray[0]));

// Print integer array through a pointer

// First print the integer array pointer set to NULL

**int** \*intArrayPtr = NULL;

**printf** ("int\* intArrayPtr %p %p %d\n", &intArrayPtr, intArrayPtr, **sizeof** (**int**\*));

// Next set pointer to point to array

intArrayPtr = intArray;

Print information for each integer array element by accessing the element through the pointer to the integer array (use pointer arithmetic)

int\* intArrayPtr 0028FF04 00000000 4 pointer set to NULL

int\* intArrayPtr 0028FF04 0028FF18 4 pointer to array

int\* intArrayPtr 0028FF18 8 4 access intArray[0] through ptr

int\* intArrayPtr 0028FF1C 1 4 access intArray[1] through ptr

int\* intArrayPtr 0028FF20 7 4 access intArray[2] through ptr

int\* intArrayPtr 0028FF24 2 4 access intArray[3] through ptr

int\* intArrayPtr 0028FF28 4 4 access intArray[4] through ptr

// Access array elements through a pointer (I removed spacing and comment to fit on line)

**printf** ("int\* intArrayPtr %p %d %d\n", &intArrayPtr[0], \*intArrayPtr, **sizeof** (**int**\*));

**printf** ("int\* intArrayPtr %p %d %d\n", &intArrayPtr[1], \*(intArrayPtr +1), **sizeof** (**int**\*));

**printf** ("int\* intArrayPtr %p %d %d\n", &intArrayPtr[2], \*(intArrayPtr +2), **sizeof** (**int**\*));

**Today’s Terminology**

**Terminology**

* C Standard Library Functions
  + Contains functions we can use to process characters, strings, lines of text and blocks of memory.

* C11 Annex K Functions
  + *Optional* annex to C standard library
  + C11 specifies many security features
  + Reduce unexpected behavior and prevent common attacks
  + Contains more secure versions of many of the functions we will see in this chapter
  + Should use if available with your compiler.
  + More on this next lecture
* Character Constants
  + **int** value represented as one character in single quotes
* String Constants (Literals)
  + Sequence of characters treated as one unit
  + Written in double quotes

**Fundamentals of Strings and Characters**

**Character Constants**

* Integer value written as one character in single quotes

**char** myCharacter = 'm';

**printf** ("myCharacter = %d\n", myCharacter);

**printf** ("myCharacter = %c\n", myCharacter);

**int** myCharacter2 = 'm';

**printf** ("myCharacter = %d\n", myCharacter2);

**printf** ("myCharacter = %c\n", myCharacter2);

**Displays**

myCharacter = 109

myCharacter = m

myCharacter = 109

myCharacter = m

**String Constants**

* Sequence of characters that end with ‘\0’
* No special type for strings in C - define with char array
* String is accessed using array name – which is pointer to 1st character – strings are basically pointers

**char** myString[] = "This is a string in C"; creates 22-character array to hold

**printf** ("myString = %s\n", myString); all characters plus ‘\0’

* Can define with pointer to char

creates the string in read-only

**const** **char** \*myStringPtr = "This is a string in C"; memory then creates a pointer with

**printf** ("myCharacter = %s\n", myStringPtr); the string’s memory address

* When using characters arrays with strings
  + **Make sure the array is big enough for string PLUS terminating character!**

**char** str[5] = "Test";

**‘\0’ is automatically inserted at the end of the string**

**Failure to understand this is common source of errors!**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| str[0] | str[1] | str[2] | str[3] | str[4] |
| ‘T’ | ‘e’ | ‘s’ | ‘t’ | ‘\0’ |

* + If not big enough – will overwrite memory following array – all kinds of issues!

**char** myString[] = "This is a string in C using char array";

**const** **char** \*myStringPtr = "This is a string in C using char pointer";

**char** stringArray[3] = "String is too big for the array";

**printf** ("myString = %s\n", myString);

**printf** ("myCharacter = %s\n", myStringPtr);

**printf** ("stringArray = %s\n", stringArray);

**Displays**

myString = This is a string in C using char array

myCharacter = This is a string in C using char pointer

stringArray = StrThis is a string in C using char array

1st 3 characters are correct but that’s it

Produces undefined results (the above was result on my system)

* + **printf** - using **%s**
    - %s causes printf to print characters until a terminating null (‘\0’)
    - If string does not have terminating null, printf prints until one is encountered in memory!
    - From above:

**char** stringArray[3] = "String is too big for the array";

**printf** ("stringArray = %s\n", stringArray);

* + - * **stringArray** has only room for 3 characters
      * Array is too small to hold string assigned to it!!!
      * Buffer overflow occurs
      * **stringArray** will not have terminating ‘\0’ character
      * printf prints value in stringArray and keeps going until it finds a ‘\0’

**Displays**

stringArray = StrThis is a string in C using char array

‘\0’ should be here This text was in the memory location

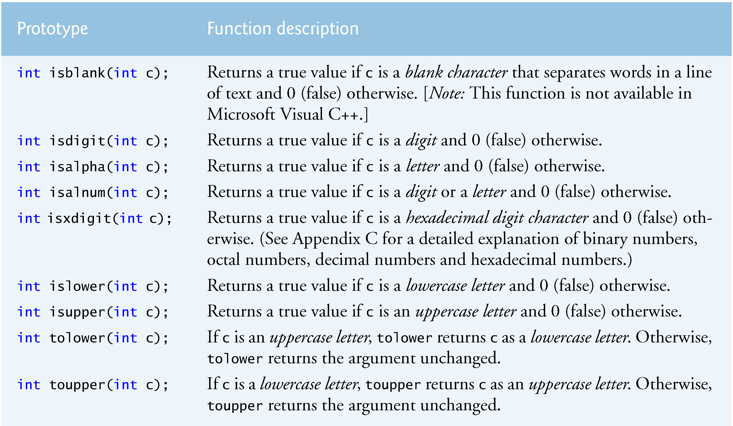
But it’s not, so issue! next to stringArray[3] so it gets printed too!

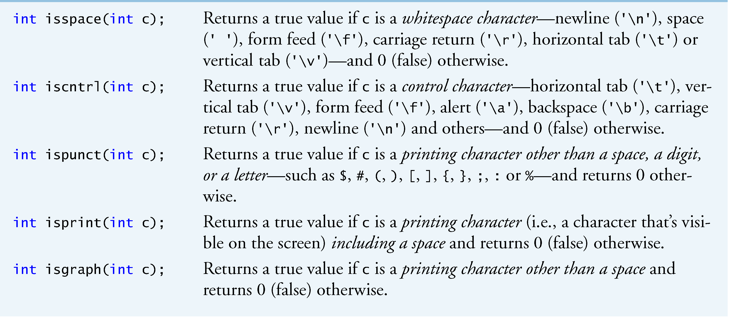
* + **scanf** - always use field width specifier with characters arrays
    - **scanf** reads until a *space, tab, newline*, *or end-of-file* indicator is reached
    - If user enters a string bigger than array size, will have issues
    - By using field width specifier, you guarantee only the max number of characters allowed are placed into the char array **plus** the terminating character
    - Without you cannot be sure if memory beyond array will be overwritten.

**Character-Handling Library**

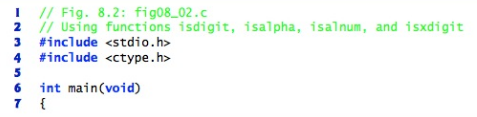
**<ctype.h>**

* Functions to manipulate characters
* Figure 8.1 is a summary of these functions
  + Notes:
    - Each function takes a int, returns an int
    - Remember characters often manipulated as integers
    - char = 1-byte integer
    - EOF normally equals -1



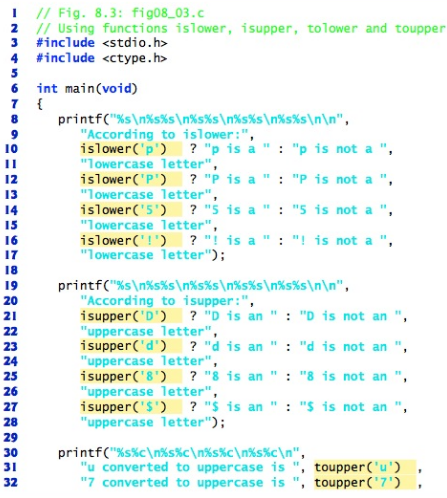


* Probably the most used
  + isDigit
  + isAlpha
  + isBlank
  + isLower
  + isUpper
  + toLower
  + toUpper
  + isSpace
* Go over code in Figure 8.2





* Go over code in Figure 8.3

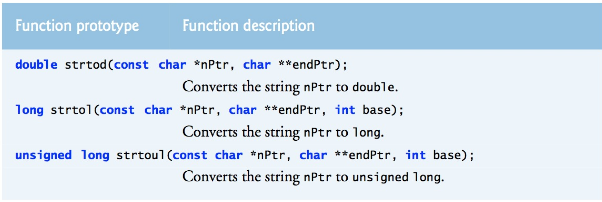




**String-Conversion Functions**

**<stdlib.h> - General utilities library**

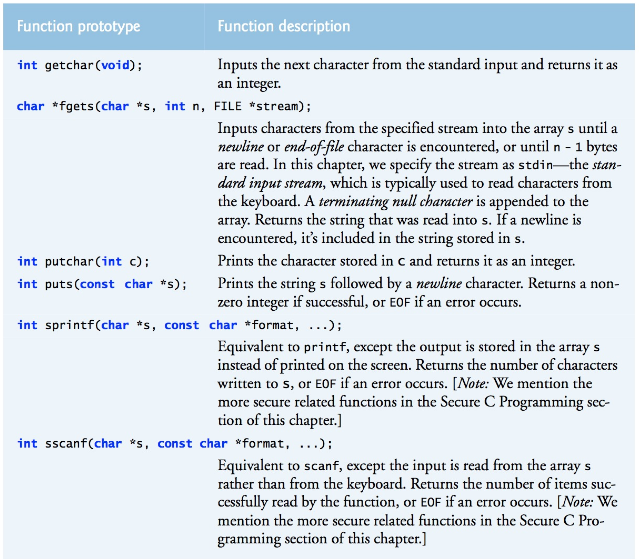
* Functions to manipulate strings
* Figure 8.5 is a summary of these functions



**Standard Input/Output Library Functions**

**<stdio.h> - Standard input/output library**

* Functions to perform input and output for characters and strings
* Figure 8.9



* **fgets** 
  + Does same thing as **scanf** but only for strings
  + **scanf** 
    - Can have buffer overflow – so use field width specifiers
    - Stops reading when encounters a *space, tab, newline,* or *end-of-file* indicator
    - Can read characters, strings, decimals, floats, etc.
  + **fgets**
    - Size is specified as a parameter – eliminates overflow issues
    - Can only read strings
    - Stops reading when encounters a *newline*, *end-of-file* indicator, or *max # characters read*
    - Handles spaces within strings – scanf stops on a space
    - Cannot take in structured data such as %d, %f
    - Better for unstructured strings

**char** lineOfText[LINE\_LENGTH];

**puts** ("Enter some text");

**fgets** (lineOfText, **sizeof**(lineOfText), stdin);

where to store read up to stream to read from

incoming data sizeof(lineOfText)-1 standard input -

leaves room for ‘\0’ keyboard

**printf** ("The text you entered = %s\n", lineOfText);

**Displays**

Enter some text

Hello there how are you

The text you entered = Hello there how are you

* Functions to read or write one character at a time
  + **getchar**
    - Reads next input character and returns it as its integer value
  + **putchar**
    - Argument is an int, returns integer variable as a character
    - Does same thing as printf and puts but **only** for one character
    - No need to provide format string
    - Some practical uses
      * Character counting
      * Word counting
      * Line counting
* Functions to store and extract formatted strings from arrays
  + **sprintf**
    - Does same thing as **printf** except to an array
      * **printf** displays output on screen
      * **sprintf** stores output in a buffer (array)
    - Gives ability to create and store formatted data in a character array
    - **snprintf** – same as sprintf but additional argument to specify number of characters
    - **snprintf\_s** - more secure version in C11

**char** array1[LINE\_LENGTH];

**sprintf** (array1,

"Placing a string with two numbers %d and %d into the array",

8, 17);

// Display what was placed into array with sprintf

**printf**("%s\n", array1);

**Displays**

Placing a string with two numbers 8 and 17 into the array

* + **sscanf**
    - Does same thing as **scanf** except from array
      * **scanf** – reads data from keyboard
      * **sscanf** – reads data from buffer (array)
    - Gives ability to extract data from character array and place in another variable
    - **sscanf\_s** - More secure version in C11

