(9-2) Strings I
H&K Chapter 8

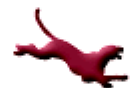
Instructor - Andrew S. O'Fallon
CptS 121 (October 16, 2024)
Washington State University

WASHINGTON STATE UNIVERSITY
World Class. Face to Face.

String Fundamentals

- A string is a sequence of characters terminated by the null character ('\0')
  - "This is a string" is considered a string literal
  - A string may include letters, digits, and special characters
- A string may always be represented by a character array, but a character array is not always a string
- A string is accessed via a pointer to the first character in it
- This week, we'll learn more about how to work with strings in the C Language

C. Hundhausen, A. O'Fallon



String Basics (1)

- Whether you realize it or not, you've been working with C strings all semester:

```
            string
printf("CptS %d is fun!\n",121);
```

- It's just that we haven't ever declared a string variable. In C, a string is represented as an array of characters:

```
char name [20]; /* declares a variable name that can hold a
                   string of length 20 */
```

- Be sure to always account for the ‘\0’ in your array declarations
  - name[ ] may have up to 19 characters + 1 for the null character

String Basics (2)

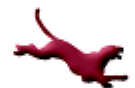
- As with other data types, we can even initialize a string when we declare it:

```
char name[20] = "Bill Gates";
char *name = "Bill Gates";
char name[] = {'B', 'i', 'l', 'l', ' ', 'G', 'a', 't', 'e',
               's', '\0';}
// These are equivalent string declarations!
```

- Here's what the memory allocated to `name` looks like after either of the above is executed:

null character (terminates all strings)

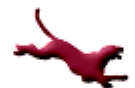
name	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	B	i	l	l		G	a	t	e	s	\0	?	?	?	?	?	?	?	?	?



C. Hundhausen, A. O'Fallon

String Basics (3)

- Notes on the null character
  - When a string is initialized on the line it is declared, the compiler automatically "null terminates" the string
  - All of C's string handling functions work only with null-terminated strings; any characters to the right of the null character are ignored
  - The ASCII value of the null character is 0



String Basics (4)

- When a variable of type `char*` is initialized with a string literal, it may be placed in memory where the string can't be modified
- If you want to ensure modifiability of a string store it into a character array when initializing it

C. Hundhausen, A. O'Fallon

String Basics (5)

- Populating a string using scanf ( )

```
char my_string [50];
// The address of operator (&) is not required because the name of the
// array is an address
scanf ("%s", my_string);
```

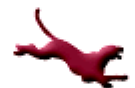
  - Notes on scanf ( ):
    - Using %s will automatically append a null character to the end of the string
    - Reads character-by-character until whitespace is encountered, i.e. if the user enters: Bill Gates, only "Bill" is read; however, "Gates" is still in the input stream

- Displaying a string using printf ( )

```
printf ("%s\n", my_string);
```

  - Notes on printf ( ):
    - Using %s will display character-by-character until a null character is encountered; white space and printable special characters will be displayed
    - If a null character is missing from the end of the string, all contiguous memory will be printed until a null character happens to be found in memory

C. Hundhausen, A. O'Fallon



String Basics (6)

- Arrays of Strings
  - Suppose we want to store a list of students in a class
  - We can do this by declaring an array of strings, one row for each student name:

```
#define NUM_STUDENTS 5
#define MAX_NAME_LENGTH 31
char student_names[NUM_STUDENTS][MAX_NAME_LENGTH];
```

  - We can initialize an array of strings "in line":

```
char student_names[NUM_STUDENTS][MAX_NAME_LENGTH] =
{"John Doe", "Jane Smith", "Sandra Connor", "Damien White",
 "Metilda Cougar"};
```

  - In most cases, however, we're probably going to want to read the names in from the keyboard or a file…

C. Hundhausen, A. O'Fallon

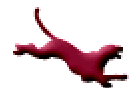
String Basics (7)

- Printing Out and Reading In Strings

```
#include <stdio.h>
#define [NUM_STUDENTS] 5
#define [MAX_NAME_LENGTH] 31
char student_names[NUM_STUDENTS][MAX_NAME_LENGTH];
int i;
for (i = 0; i < NUM_STUDENTS; ++i)
{
  printf("Please enter student name: ");
  scanf("%s",student_names[i]);
  printf("The name '%s' was just read in.\n",
               student_names[i]);
}
```

- Is the above code robust? Could it lead to a run-time crash?

C. Hundhausen, A. O'Fallon



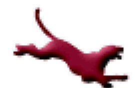
String Basics (8)

- Just as is the case for `double`s and `int`s, we can specify a field width in a `printf` statement involving a string (`%s`). By default, the string is right justified within that field, e.g.,

```
printf("string value: %5s\n",my_string);
/* string is right justified within field of 5 */
```

- If we want to left-justify the string, we specify a *negative* field width, e.g.,

```
printf("string value: %-5s\n",my_string);
/* string is left justified within field of 5 */
```

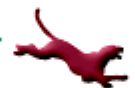
String Basics (9)

- Reading in multiple data types alongside the string data type:

```
1.  #include <stdio.h>
2.
3.  #define STRING_LEN  10
4.
5.  int
6.  main(void)
7.  {
8.        char dept[STRING_LEN];
9.        int  course_num;
10.       char days[STRING_LEN];
11.       int  time;
12.
13.       printf("Enter department code, course number, days and ");
14.       printf("time like this:\n> COSC 2060 MWF 1410\n> ");
15.       scanf("%s%d%s%d", dept, &course_num, days, &time);
16.       printf("%s %d meets %s at %d\n", dept, course_num, days, time);
17.
18.       return (0);
19. }

Enter department code, course number, days and time like this:
> COSC 2060 MWF 1410
> MATH 1270 TR 800
MATH 1270 meets TR at 800
```


C. Hundhausen, A. O'Fallon



String Basics (10)

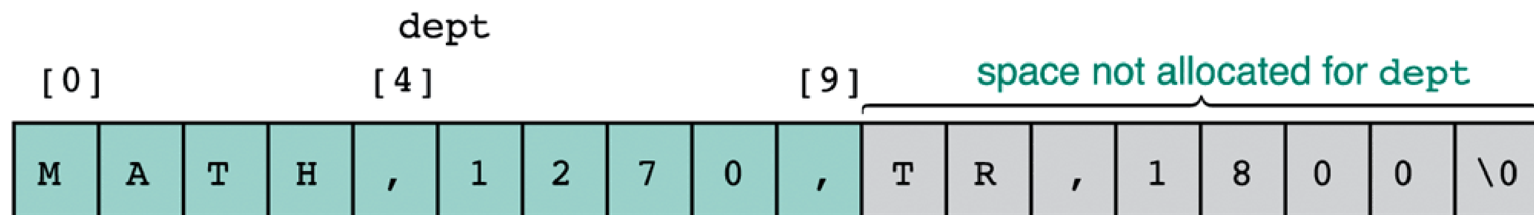
- When the previous program is run and the user enters the following (which is not in the correct format):

```
MATH,1270,TR,1800
```

The `scanf` call

```
scanf("%s%d%s%d",dept,&course_num,days,&time);
```

interprets this all as one string, storing it to `dept` (bad news!):



Moral: We need a more robust way to read in multiple data types (Stay tuned!)

C. Hundhausen, A. O'Fallon

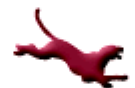
String Basics (11)

- Example problem:
  - Write a segment of code that prompts the user for a word of length 24 characters or less, and prints a statement like this:

```
fractal starts with the letter f
```

    Have the program process words until it encounters a "word" beginning with the character `'9'`.

C. Hundhausen, A. O'Fallon

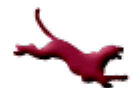


String Basics (12)

- Solution:

```
#include <stdio.h>
#define STRING_LENGTH 25

int main()
{
  char name[STRING_LENGTH];
  int done;
  do
  {
     done = 0;
     printf("Enter a name ('9') to quit: ");
     scanf("%s",name);
     if (name[0] == '9')
       done = 1;
     else
       printf("%s starts with the letter %c.\n",
              name,name[0]);
   } while (!done);
   return (0);
}
```

String Basics (13)

- Use `gets()` to read a complete line, including whitespace, from the keyboard until the <enter> key is pressed; the <enter> is not included as part of the string
  - Usage: `gets(my_array)`
  - If the user enters "Bill Gates" and presses <enter>, the entire string will be read into `my_array` excluding the <enter> or newline
- Use `puts()` to display a string followed by a newline
  - Usage: `puts(my_array)`

C. Hundhausen, A. O'Fallon

What To Look Forward To…

- More on Strings:
  - String handling library functions
  - Arrays of Pointers
  - Character input/output and robust string input
  - Character conversion
  - String processing example

References

- J.R. Hanly & E.B. Koffman, *Problem Solving and Program Design in C (8th Ed.)*, Addison-Wesley, 2016.
- P.J. Deitel & H.M. Deitel, *C How to Program (7th Ed.)*, Pearson Education , Inc., 2013.

Collaborators

- Chris Hundhausen