Strings

Hour 13

- Objectives
  - Defining strings: character arrays and character pointers
  - String I/O: printf, scanf, gets, fgets, puts, and fputs
  - String functions
  - In memory formatting: sprintf and sscanf

The two string representations

- Null-terminated single-dimensioned character array
  - Array must be 1 character longer than the longest string that it holds
  - Compiler automatically inserts the null at the end of string constants
  - The array name, without brackets, is a constant character pointer

- Character pointer; before use, it should point
  - to a character array
  - to a string constant

```c
char s1[100] = "Example";
char* s2 = s1;
char* s3 = "Hello";
```
String IO with `printf` and `scanf`

### Standard I/O

- `char* message = "hello world";`
- `char buffer[100];`
- **`printf`**
  - use the `%s` format specifier
  - can display either a char* or a char[]
  - `printf("%s %s\n", message, buffer);`
- **`scanf`**
  - use the `%s` format specifier
  - must read into a character array; array name is an address
  - `scanf("%s", buffer);`
  - doesn’t read through spaces: i.e., reading “hello world” only gets hello

---

String Functions

### ANSI string library functions

- **C/C++ does not have intrinsic string manipulation operators**
  - Operations are done by functions contained in the run-time library
  - Prototypes, etc. are in `<string.h>`
  - Assumes that all strings are null-terminated
  - Do “man string” for more information
- **Most useful functions**
  - `int strlen(char* s)` /* length; does not include null */
  - `int strcmp(char* s1, char* s2)` /* returns < 0, 0, > 0 */
  - `char* strcpy(char* s1, char* s2)` /* s1 ← s2 */
  - `char* strdup(char* s)` /* duplicates, s returns pointer */
  - `char* strcat(char* s1, char* s2)` /* appends s2 at the end of s1 */
  - `char* strchr(char* s, char c)` /* search for char c in string s */
  - `char* strstr(char* s1, char* s2)` /* search for string s2 in string s1 */
  - `char* strtok(char* s1, char* s2)` /* tokenize s1; delimiters in s2 */
Alternate String Example

Focusing on addresses

```c
char string[8] = "EXAMPLE";
char* strptr = string; /* array name is an address */

printf("%s %s\n", string, strptr);
printf("%d   %d\n", strlen(string), strlen(strptr));
```

<table>
<thead>
<tr>
<th>string</th>
<th>ExAMple</th>
</tr>
</thead>
<tbody>
<tr>
<td>strptr</td>
<td>100 108</td>
</tr>
</tbody>
</table>

Hour 13 C Slide 5 of 12

String Input

Reading one line at a time

- `char* fgets(char* buf, int n, FILE* fp)`
  - Reads either n-1 characters or until new-line (whichever is shortest)
  - Stores the new-line if read
  - Always null-terminates the line
- `char* gets(char* buf) /* not recommended */`
  - Reads until new-line and discards the new-line
  - No test to prevent reading beyond the end of the buffer
- Returns NULL at end of file or error, buf otherwise
- Common idiom:
  ```c
  char line[256];
  while (fgets(line, 256, fp) != NULL)
  { line[strlen(line)-1] = '\0';
    /* process line */
  }
  ```

Hour 13 C Slide 6 of 12
String Output
Writing one line at a time

- int fputs(const char* str, FILE* fp)
- int puts(const char* str)
- Returns EOF on error, non-negative value otherwise
- str must be null-terminated, but the null is not written out

sprintf And sscanf
In-core formatting

- #include <stdio.h>
- “in-core formatting” or “in memory formatting”
- int sprintf(char *buffer, “controlString”, arg1, arg2, ...);
  ▶ The control string and conversions are like printf
  ▶ “Output” is written to buffer (a character array)
  ▶ Used to dynamically build output messages
- int sscanf(char *buffer, “controlString”, arg1, arg2, ...);
  ▶ The control string and conversion are like scanf
  ▶ “Input” is taken from buffer
  ▶ All arguments must be addresses
  ▶ Used for data input validation
Comparing Strings

Are two strings equal?

- `strcmp(char* s1, char* s2)`
  - returns a value < 0 if `s1` is ordered before `s2`
  - returns 0 if `s1` and `s2` are identical
  - returns a value > 0 if `s1` is ordered after `s2`
  - is case-sensitive
  - short strings sort before long strings: `strcmp("aa", "aaa") < 0`

```
strcmp("hello", "world") is -15 104 ('h') - 119 ('w') = -15
strcmp("world", "hello") is  15 119 ('w') - 104 ('h') =  15
strcmp("hello ", "hello ") is   0 'h' - 'h' = 0; 'e' - 'e' = 0;
   'l' - 'l' = 0; 'l' - 'l' = 0;
   'o' -'o' = 0
```

Concatenating Strings

The `strcat` function

```c
char  s1[100] = "Hello";
char* s2 = " world";
strcat(s1, s2);
```

```c
char* strcat(char* s1, char* s2)
{
    int len = strlen(s1);
    for (int i = 0; i <= strlen(s2); i++)
        s1[len + i] = s2[i];

    return s1;
}
```
**String Function Examples**

The `strchr` and `strstr` functions

```c
s = hello world\0 ....
```

- `s1 = strchr(s, 'e');`
- `s2 = strchr(s, 'a');`
  - `s2` is `NULL`
- `s3 = strstr(s, "world ");`
- `s4 = strstr(s, "fred ");`
  - `s4` is `NULL`

- `printf("%s\n ", s1);` prints: `ello world`
- `printf("%s\n ", s3);` prints: `world`

**Write The `strcpy` Function**

An example based on arrays

```c
char *strcpy(char *s1, const char *s2) {
  int i;
  for (i = 0; i <= strlen(s2); i++)
    s1[i] = s2[i];
  return s1;
}
```

```c
char* strcpy(char* s1, const char* s2) {
  int i = 0;
  while ((s1[i] = s2[i]) != '\0')
    i++;
  return s1;
}
```