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

Strings

The two string representations

- Null-terminated single-dimensioned character array
  - Array must be 1 character longer than the longest string 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

```
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("%s\n", message, buffer);
- scanf("%s", buffer);

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, > */
  - char* strcpy(char* s1, char* s2) /* s1 \² 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", strlen(string), strlen(strptr));
```

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 end of the buffer
```

```
Returns NULL at end of file or error, buf otherwise
```

```
Common idiom:
  char line[256];
  while ((fgets(line, 256, fp) != NULL)
  { line[strlen(line)-1] = '\0';
    /* process line */
  }
```
### String Output

Writing one line at a time

```c
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

```c
#include <stdio.h>
```

`in-core formatting` or “in memory formatting”

```c
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

```c
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?

```c
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
- `strcmp("world", "hello")` is 15
- `strcmp("hello ", "hello ")` is 0

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

### Concatenating Strings

The `strcat` function

```c
char* strcat(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`

### String Function Examples

The `strchr` and `strstr` functions

```c
char* strchr(char* s, 'e');
char* strstr(s, "world ");
s1 = strchr(s, 'e');
s2 = strchr(s, 'a');
s2 is NULL
```

```c
printf("%s\n", s1);
printf("%s\n", s3);  
```

### Write The `strcpy` Function

An example based on arrays

```c
char* strcpy(char* s1, const char *s2)
```

```c
for (i = 0; i <= strlen(s2); i++)
return s1;
```