Strings

Chapter 9

CStrings

Two character-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

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

String Input

Reading one line at a time

```c
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, but otherwise

Common idiom:
```c
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)
- Returns EOF on error, non-negative value otherwise
- str must be null-terminated, but the null is not written out
```

`printf` And `sscanf`

In-core formatting

```c
#include <stdio.h>

"in-core formatting" or "in memory formatting"

int printf(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
```
String Functions

ANSI string library functions

- 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

- Most useful functions
  - `strlen(char* s)` /* length; does not include null */
  - `strcmp(char* s1, char* s2)` /* returns < 0, 0, > 0 */
  - `char* strcpy(char* s1, char* s2)` /* s1 = s2 */
  - `char*strupr(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 */

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 0 111 ("h") - 111 ("h") = 0; \"e\" - \"e\" = 0;
\'l\' - \'l\' = 0; \'l\' - \'l\' = 0;
\'o\' - \'o\' = 0
```

Concatenating Strings

The `strcat` function

```
char s[100] = "Hello";
char* strcat(char* s1, char* s2)
{
  int len = strlen(s1);
  strcat(s1, s2);
  for (int i = 0; i <= strlen(s2); i++)
    s1[len + i] = s2[i];
  return s1;
}
```

Write The `strcpy` Function

An example based on arrays

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

Character Identification and Mapping

- Using ctype macros
  - Macros return 1 for true and 0 for false
  - isascii() is defined on all integer values, the rest are defined only for integers representing characters, or EOF
  - #include <ctype.h>
  - int ispunct(int c)
  - int isalpha(int c)
  - int isupper(int c)
  - int islower(int c)
  - int isdigit(int c)
  - int isxdigit(int c)
  - int isalnum(int c)

Command-Line Arguments

Array of c-strings

```c
int main(int argc, char* argv[]) {
    c:\> argdemo hello world from CS2250
    argv[]
    argc 5
    for (i = 1; i < argc; i++)
        process(argv[i]);
    for (i = 1; argv[i] != NULL; i++)
        process(argv[i]);
    for (i = 1; argc[i]; i++)
        process(argv[i]);
    NULL
}
```