String Functions

2024 Winter APS 105: Computer Fundamentals

Jon Eyolfson

Lecture 23

1.0.1

There are String Functions in the Standard Library

We just need to `#include <string.h>` at the top of our code

Remember: C strings are arrays of characters that end with a null byte

Some of these functions are tricky to use correctly,
we need string length + 1 bytes to store a C string

Can Use a Function to get the Length of a String

Instead of writing it ourselves, we can use:

```
  size_t strlen(const char *s);
```

returns the length of the string (you could explicitly cast a `size_t` to an `int`)

`s` is a null terminated string

Note: you may hear people say null terminated string, or “C string”,
  they mean the same thing

We Can Get the Length of a String with `strlen`

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

int main(void) {
    const char *s = "Hello world";
    int sLen = (int) strlen(s);
    printf("strlen(s): %d\n", sLen);
    return EXIT_SUCCESS;
}
```

Maybe We're Not Sure the String is Null Termianted

We can limit the number of characters the function reads with:

```
  size_t strnlen(const char *s, size_t maxlen);
```

This function only accesses `s[0]` to `s[maxlen - 1]` even if there's no null byte

The maximum value `strnlen` returns is `maxlen` even if `s` is a longer C string

Limit the Maximum Length with `strnlen`

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

int main(void) {
    const char *s = "Hello world";
    int sLen = (int) strnlen(s, 5);
    printf("strnlen(s, 5): %d\n", sLen);
    return EXIT_SUCCESS;
}
```

We'll see:

```
  strnlen(s, 5): 5
```

We Can Use `strcpy` to Copy a String

Remember, we can't modify a string if we do:

```
  char *s = "Hello world";
```

We could copy a string to the heap with:

```
  char *strcpy(char *dest, const char *src);
```

returns the same address as `dest`

`src` is the C string to copy values from

`dest` is the location to copy values to in memory

- `dest` needs to point to (at least) `strlen(src) + 1` bytes of valid memory
- `dest` will be null terminated

Copying a String with `strcpy`

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

int main(void) {
    const char *s = "Hello world";
    char *d = malloc(sizeof(char) * 5);
    strcpy(d, s); /* d is not large enough! */
    printf("s: %s\n", s);
    printf("d: %s\n", d);
    return EXIT_SUCCESS;
}
```

Copying a String with Limits

We can use this function to copy instead:

```
char *strncpy(char *dest, const char *src, size_t n);
```

This functions always writes `n` characters to `dest`

- In other words, it assigns values from `dest[0]` all the way to `dest[n - 1]`

Copying a String with Limits

We can use this function to copy instead:

```
char *strncpy(char *dest, const char *src, size_t n);
```

This functions always writes `n` characters to `dest`

- In other words, it assigns values from `dest[0]` all the way to `dest[n - 1]`

If `n < (strlen(src) + 1)` then `dest` is **not** null terminated

If `n > (strlen(src) + 1)` then `dest` is filled with null bytes

- for values from `dest[strlen(src)]` to `dest[n - 1]`

Copying a String with `strncpy`

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

int main(void) {
    const char *s = "Hello world";
    printf("s: %s\n", s);
    int size = 6;
    char *d = malloc(sizeof(char) * size);
    strncpy(d, s, size);
    /* There's no overflow, but d does not have a null byte */
    if (d[size - 1] != '\0') {
        printf("There's no null byte!\n");
        d[size - 1] = '\0';
    }
    printf("d: %s\n", d);
    free(d);
    return EXIT_SUCCESS;
}
```

We Can Extend a String with the Content of Another

In other words, we can concatenate a string with:

```
char *strcat(char *dest, const char *src);
```

This copies the `src` C string to the end of `dest`

- It starts copying `src` from the null byte of `dest`

`dest` must be able to hold `strlen(src) + strlen(dest) + 1` bytes

Copying a String with `strcat`

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

int main(void) {
    const char *s = " world";
    int size = 12;
    char *d = malloc(sizeof(char) * size);
    strncpy(d, "Hello", size);
    strcat(d, s);
    puts(d);
    return EXIT_SUCCESS;
}
```

Example Implementation of `strcat`

```c
char* strcat(char *dest, const char *src) {
    size_t dest_len = strlen(dest);
    size_t i;
    for (i = 0; src[i] != '\0'; ++i) {
        dest[dest_len + i] = src[i];
    }
    dest[dest_len + i] = '\0';
   return dest;
}
```

We Can Limit the Number of Characters We Copy

Similar to copying a string, we can set limits with:

```
char *strncat(char *dest, const char *src, size_t n);
```

This copies at most `n` bytes from `src`

- `src` does not need to be null terminated

The function also always adds a null byte to the end of `dest`

- `dest` must point to `strlen(dest) + n + 1` bytes of valid memory

strncat Should Only Write the Number of Bytes that Fit

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

#define ARRAY_LENGTH(array) (sizeof((array))/sizeof((array)[0]))

int main(void) {
    const char s[] = {' ', 'J', 'o', 'n'};
    int size = 10;
    char *d = malloc(sizeof(char) * size);
    strncpy(d, "Hello", size);
    /* Still doesn't take into account the number of bytes in d */
    strncat(d, s, ARRAY_LENGTH(s));
    /* This is better. */
    // strncat(d, s, size - strlen(d) - 1);
    puts(d);
    return EXIT_SUCCESS;
}
```

We Can Compare Strings with Each Other

We need to use the function:

```
  int strcmp(const char *s1, const char *s2);
```

returns an integer,

- a negative value means `s1 < s2`
- a 0 value means `s1 == s2`
- a positive value means `s1 > s2`

We Can Compare Strings with Each Other

We need to use the function:

```
  int strcmp(const char *s1, const char *s2);
```

returns an integer,

  a negative value means `s1 < s2`
  a 0 value means `s1 == s2`
  a positive value means `s1 > s2`

Note: we need to use this function to compare strings,
  otherwise we're comparing addresses (pointers)

`strcmp` Compares Both Strings Character by Character

It uses the ASCII values, and stops at the first character that differs:

- `strcmp("9", "10")` → positive, which means `"9" > "10"`
- `strcmp("jo", "jon")` → negative, which means `"jo" < "jon"`
- `strcmp("aaa", "b")` → negative, which means `"aaa" < "b"`

Comparing Strings Using `strcmp`

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

int main(void) {
    const char *s1 = "alpha";
    const char *s2 = "beta";
    if (strcmp(s1, s2) < 0) {
        printf("s1 is less than s2!\n");
    }
    else if (strcmp(s1, s2) > 0) {
        printf("s1 is greater than s2!\n");
    }
    else {
        printf("s1 and s2 are the same!\n");
    }
    return EXIT_SUCCESS;
}
```

The Pointers Differ, but the String Content is the Same

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

int main(void) {
    const char* s = "Testing";
    int sLen = (int) strlen(s);
    char *s1 = malloc(sizeof(char) * (sLen + 1));
    char *s2 = malloc(sizeof(char) * (sLen + 1));
    strncat(s1, s, sLen + 1);
    strncat(s2, s, sLen + 1);
    if (s1 == s2) { printf("Pointers are the same\n"); }
    else          { printf("Pointers are DIFFERENT\n"); }
    free(s1);
    free(s2);
    return 0;
}
```

We Can Only Compare with Limits

We change use the function:

```
  int strncmp(const char *s1, const char *s2, size_t n);
```

This function compares at most `n` bytes from both strings

The strings do not have to be null terminated in this case,
 it will stop reading early when a null byte occurs in either string

Comparing Only the First 3 Characters of Two Strings

```c
#include <string.h>
#include <stdlib.h>

int main(void) {
    const char *s1 = "joneyolfson";
    const char *s2 = "jonstewart";
    int result = strncmp(s1, s2, 3);
    if (result < 0) {
        printf("s1 is less than s2!\n");
    }
    else if (result > 0) {
        printf("s1 is greater than s2!\n");
    }
    else {
        printf("s1 and s2 are the same!\n");
    }
    return 0;
}
```

We Can Search for a String within a Larger String

We can use this function that searches for a substring:

```
  char *strstr(const char *haystack, const char *needle);
```

returns a pointer to the starting character of `needle` in `haystack`
  or `NULL` if it does not exist

Note: the returned (non-`NULL`) pointer will point to a character in `haystack`

Using strstr to Find the Word “long”

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

int main(void) {
    const char *s = "This is a long line!";
    const char *f = strstr(s, "long");
    if (f != NULL) {
        printf("f: %s\n", f);
    }
    return EXIT_SUCCESS;
}
```

This will print:

```
  f: long line!
```

We Can Also Find a Character in a String

We can use the following function instead:

```
  char *strchr(const char *s, int c);
```

returns a pointer to the character in `s`
   or `NULL` if the character does not exist

Using strchr to Find the First “w”

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

int main(void) {
    const char *s = "Hello world";
    const char *f = strchr(s, 'w');
    if (f != NULL) {
        printf("f: %s\n", f);
    }
    return EXIT_SUCCESS;
}
```

This will print:

```
  f: world
```

We Can Convert Strings to an `int` or `double`

Both of these functions are in `stdlib.h`:

```
  int atoi(const char *s);
  double atof(const char *s);
```

`atoi` converts the string `s` to an `int` and returns its value

`atof` converts the string `s` to a `double` and returns its value

An Example of Using `atoi`

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

int main(void) {
    int value = atoi("1234");
    printf("value: %d\n", value);
    return EXIT_SUCCESS;
}
```

An Example of Using atof

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

int main(void) {
    double value = atof("1.337");
    printf("value: %lf\n", value);
    return EXIT_SUCCESS;
}
```

Summary of String Functions

```
#include <string.h>
  size_t strlen(const char *s);
  size_t strnlen(const char *s, size_t maxlen);
  char *strcpy(char *dest, const char *src);
  char *strncpy(char *dest, const char *src, size_t n);
  char *strcat(char *dest, const char *src);
  char *strncat(char *dest, const char *src, size_t n);
  int strcmp(const char *s1, const char *s2);
  int strncmp(const char *s1, const char *s2, size_t n);
  char *strstr(const char *haystack, const char *needle);
  char *strchr(const char *s, int c);

#include <stdlib.h>
  int atoi(const char *s);
  double atof(const char *s);
```