L16 - Strings

November 27, 2019

1 Strings

Administrative stuff

- no class on Dec 4
- Dec 6 has Monday schedule
- final weighted toward `malloc()` stuff

Unlike Python and C++, C does *not* have a string type. Strings are implemented as arrays of characters, terminated with a `\0` or *null character*. Upon declaration, the C compiler will create *and initialize* the array.

Note that the null character is not the same as the "0" character or the `NULL` null pointer.

The number of elements in a character array is one more than the number of characters it is initialized with, to accomodate the null character.

C does not allow assinging a string literal into a character array.

As with other variables, the `const` keyword tells the compiler that the variable is *immutable*. Attempting to write into or modify a `const`'d variable will generate a compilation error.

C operators are not overloaded to allow for string concatenation - you cannot `+` two strings together.

1.1 `strlen()`

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

Returns the length of a string pointed to by s, not including the terminating null. `size-T` is controlled by the compiler in use; it is always some unsigned integer type, but might be short, long, long long, etc.

```
In [36]: #include <string.h>
         #include <stdio.h>

         void main()
         {
             char greeting[] = "Hello";
             size_t len;
             len = strlen(greeting);
             // returns 5 (not 6)
             printf("%zd", len); // use 'z' modifier to turn size_t to int, I guess
         }
```

```
5
```

1.2 strcmp()

`int strcmp(const char *s, const char *t)`

Compares the string pointed to by s to the string pointed to by t. Specifically, it returns whether the sum of the ASCII encodings in s is greater than t.

Returns a positive value if `s > t`, zero if `s == t`, and a negative value if `s < t`. A useful form would be to use `!= 0` to find out if two strings are different:

```
In [ ]: char name1[30];
        char name2[30];

        void main()
        {
            // Initialization of name1 and name2
            // not shown
            if (strcmp(name1, name2) != 0) {
            // strings are different
            }
        }
```

1.3 strstr()

`char *strstr( const char *s, const char *t )`

Searches for the first occurence of the string pointed to by `*t` *within* the string pointed to by `*s`. If the string is found, it returns a pointer to the located string. If not, it returns `NULL`.

If you just need to know if the `*t` is in there, just use `== NULL`.

1.4 strcpy()

`char *strcpy ( char *dst, const char *src )`

Copies all characters in `*src` to `*dst` (including the null terminator). Returns `dst`. **You must ensure `dst` is big enough to hold all the characters that are in `src`.**

1.5 strcat()

`char *strcat( char *dst, const char *src )`

Appends a copy of the string (including the null terminator) that is in `*src` to the string that is in `*dst`. `*src`'s null terminator is overwritten by the first character in `*dst`. **You must ensure `dst` is big enough to hold all the characters that are in `src`.**

1.6 sprintf()

Similar to `printf()`, but stores the output in the first argument. Note that this will *not* output to the console.

1.7 An implementation of strlen()

```
In [4]: #include <stdio.h>
        #include <string.h>
```

```c
#include <stdlib.h>

// a DIY strlen by array indexing
size_t indexing_strlen(const char s[])
{
    size_t i = 0;
    while (s[i] != '\0')
    {
        i += 1;
    }
    return i; // doesn't include '\0'
}

// with a walking pointer
size_t walking_strlen(char* s)
{
    size_t n = 0;
    while ( *s != '\0' )
    {
        n++;
        s++;
    }
    return n;
}

void main()
{
    char msg1[] = "SYSC2006 2019";
    size_t result1, result2;

    result1 = indexing_strlen(msg1);
    printf("array indxing implementation: %zu\n", result1);

    result2 = walking_strlen(msg1);
    printf("walking pointer implementation: %zu\n", result2);


}
```

```
array indxing implementation: 13
walking pointer implementation: 13
```

1.8 An implementation of `strcmp()`

In [ ]:
```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

```
int CU_strcmp(const char s[], const char t[])
{
int i;
for (i = 0; s[i] == t[i]; i = i + 1) {
if (s[i] == '\0') {
// reached the end of both strings;
// all chars are equal
return 0;
}
}
// i is first position where s and t differ
return s[i] - t[i];
}

int CU_strcmp(const char *s, const char *t)
{
while (*s == *t) {
if (*s == '\0') {
// reached the end of both strings;
// all chars are equal
return 0;
}
s += 1;
t += 1;
}
return *s - *t;
}
```

In [17]:
```
#include <string.h> // library for fun string stuff
#include <stdio.h>
#include <stdlib.h>

void main()
{
    const char str1[] = "SYSC2006";
    const char str2[] = "F19";

    printf("%s %s\n", str1, str2);

    char msg[50];

    sprintf(msg, "%s %s", str1, str2);

    printf("%s\n", msg);
}
```

```
SYSC2006 F19
```

1.9 sscanf()

Reads from the first argument, which should be a string. Can be used to parse function arguments.

```
In [23]: #include <string.h>
         #include <stdio.h>
         #include <stdlib.h>

         void main()
         {
             char msg1[] = "SYSC2006 2019";

             printf("msg1 = %s\n", msg1);

             int year;
             char str1[22];

             sscanf(msg1, "%s %d", str1, &year);

             printf("str1 = %s, year = %d\n", str1, year);
         }
```

```
msg1 = SYSC2006 2019
str1 = SYSC2006, year = 2019
```

1.10 TCP/IP Network Protocol Emulation

A 4-tuple gets attached to a packet on the network: * source IP address

- destination IP address
- source port number
- destination port number

```
In [26]: #include <string.h>
         #include <stdio.h>
         #include <stdlib.h>

         // Encode a message ( mostly from the slides )

         char src[] = "10.1.1.1"; // Src IP addr
         char dst[] = "10.2.2.2"; // Dest IP addr
         int tos = 4; // Type of Service
         int msgLen = 200; // packet length
```

```
        void main()
        {
            // somewhere to put our message
            char msg[100];

            // semicolon separated string thing
            sprintf(msg, "%s;%s;%d;%d", src, dst, tos, msgLen);

            printf("msg = %s\n", msg);
        }
```

```
msg = 10.1.1.1;10.2.2.2;4;200
```

In [32]:
```
// Decode (parse) a message ( also mostly from the slides )
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

char src[20];
char dst[20];
int tos;
int len;

// parse the message into several variables
void main()
{
    // a message to decode
    char msg[] = "10.1.1.1;10.2.2.2;4;200";

    sscanf(msg, "%[^;];%[^;];%d;%d", src, dst, &tos, &len);

    printf("src = %s, dst = %s, tos = %d, len = %d\n\n", src, dst, tos, len);
}
```

```
src = 10.1.1.1, dst = 10.2.2.2, tos = 4, len = 200
```