ECE 471 – Embedded Systems
Lecture 24

Vince Weaver

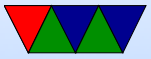
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

30 October 2023

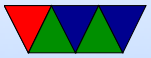
Announcements

- HW#8 was posted
- Remember project ideas due soon



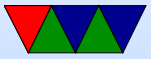
HW#8 – C string review

- String manipulation is famously horrible in C.
- There are many ways to get the "YES" and "t=24125" values out of the text file for HW#8.
- Any way you choose is fine.



C String Review

- This is tricky to get right
- It's relevant to Computer Security, the next topic we will cover

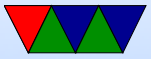


What is a C string? – essentially a hack

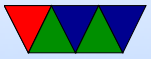
- A NUL (zero) (note: not NULL) terminated array
H	e	l	l	o	\0
- Note this is really:

0x48	0x65	0x6c	0x6c	0x6f	0x00
- Note in C, arrays are essentially just pointers
- Can statically declare: (compiler puts the 0 on end for you)

```
char string1[6]="Hello";
char string1[]="Hello";  // autosize
```

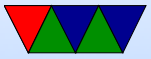


```
char *string2="Hello";
```



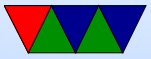
C String Review

- Many issues with array of bytes vs string, especially in other languages. Complicated if Unicode or UTF8. Windows / java and wchar (16-bit chars)
- You can use either pointer or array access to get a value (string[0] is the same as *string)
- Note that double quotes indicate a string, while single quotes indicate a single character



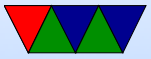
Upsides of C strings

- Fast and simple to deal with in assembly language
- Can quickly make short and cryptic functions to manipulate them
- ???



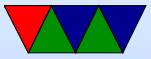
Downsides of C strings

- No way to tell the maximum size from the pointer
- Can only find out current size of string by iterating to find end
- The C library has a lot of helper functions, many of which are flawed in deep ways



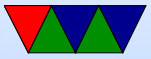
Other String Implementations

- Pascal-style strings, first byte is the length
  - Always know length, no need to strlen()
  - Maximum size (if 8-bit than max 256 chars)
- Higher level / object oriented languages (python, C++?) still have some sort of array of chars inside, but wrap it with extra info to provide safer access to things



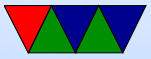
C string pitfalls – Writing off the End

- What happens when web form on your device's web interface asking "name" and you allocate 64 bytes but don't check, and someone types 4096 bytes
- What's the worst case?
- Crash your program?
- Corrupt data?
- Complete system compromise?

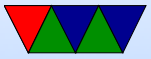


Can the C-library string functions save you?

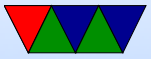
- The standard strcpy(char *dst, char *src)
  - will happily go off the end if destination smaller than source
- strncpy(char *dst, char *src, int size)
  - added destination-size parameter, also pads dest with 0
  - NOTE: will leave off (!) the NUL terminator if not fit
- strlcpy(char *dst, char *src, int size)
  - always terminates destination



- if destination full, you lose a byte as it is silently truncated and last byte made NUL
- No error is indicated if this happens
- why a problem? example: say want to remove file.txt but got got truncated to file.txt instead?
- `https://lwn.net/Articles/507319/`

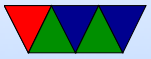


HW#8 Challenge – Reading from File



Method One – File I/O Using fscanf()

- The "stream" file interface in C lets you used buffered I/O and is slightly higher level than `open()`/`close()`
- Open a file with: `FILE *fff;`
  `fff=fopen("filename","r");`
  Check for errors! fff==NULL if it fails to open
- close a file with `fclose(fff);`
- you can read a string using `fscanf(fff,"%s",string);`

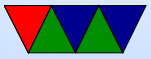


notes on scanf() functions

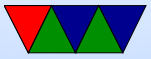
- printf() like interface

```
char string[256];
int x;
scanf("%d %s",&x,string);
```

  - Types to read like in printf, d for integer, s for string
  - Useful trick, %*s the asterisk means read but don't output, useful for skipping things
  - Result goes to a pointer. Note a string is already a pointer so no need for an ampersand

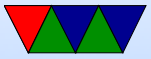


- scanf() reads from standard input (keyboard)
- fscanf() reads from file
- sscanf() reads from a string



Method Two – Read Entire File into RAM

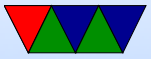
- There are multiple ways to read files into a string in C
  Assume `char string[1024];`
  - `fd=open("filename",RD_ONLY);`
    `result=read(fd,string,1023); close(fd);`
  - `FILE *fff; fff=fopen("filename","r");`
    `fread(buffer,size,count,fff); fclose(fff)`
- If you are treating things as a string, be sure to NUL-terminate `string[result]=0;`



Hardcoded sizes

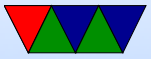
- In the last example I was being lazy and hardcoded a 1k size instead

  Can you make that dynamic?
- Use `stat()` to get filesize, then use `malloc()` to allocate space? Be sure to `free()` when done



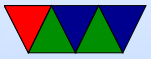
Other ways to access file contents

- Advanced: use `mmap()`
- You can also use `fgets(buffer,size,fff);` to bring in one line at a time
- What about `gets()`? Dropped from C libraries as being too unsafe! No size so just writes forever

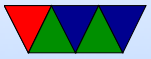


Finding a location / substring in a larger string

- If you trust the Linux kernel developers to keep a "stable ABI" you can assume the temperature will always be a fixed offset and hard code it. This can be a bit dangerous.
- You can use the `scanf()` series of functions to parse the string (either fscanf() directly, or sscanf() on the string) One helpful hint, putting a '*' in a conversion (like `%*s` tells scanf to read in the value but ignore it.

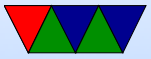


- You can use the `strstr()` search for substring C-library function to search for substrings, i.e. `strstr(string,"NO");` (haystack, needle)
- Maybe in conjunction with `strtok()`?
- You can manually parse the array.
  Using array syntax, something like:
  ```
  i=0; while(string[i]!=0) {
  if (string[i]=='t') break; i++ }
  ```
  Using pointer syntax, something like:
  ```
  char *a; a=string; while(*a!=0) {
  if (*a=='t') break; a++; }
  ```



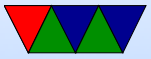
Pointing into a string

- If you searched for "t=" you might now have a pointer a to something like "t=12345". To point to 12345 you can just add 2 to the string pointer.
- `printf("%s\n",string+2);`
- `printf("%s\n",&string[2]);`



Converting string to decimal or floating point

- `atoi(char *string)` converts string to integer. What happens on error?
- `strtol()` will give you an error but is more complex to use
- `atof()` and `strtod()` will do floating point



Comparing strings

- Can you just use ==? NO!
- Be careful using `strcmp()` (or even better, `strncmp()`
  they have unusual return value
  less than, 0 or greater than depending. 0 means match
  So you want something like
  `if (!strcmp(a,b)) do_something();`