

CSE 29

Lecture 6 Summary

January 22, 2026



Logistical Things

- Assignment 2 is due next Thursday (1/29), which includes:
 - PSet 2 on Prairie Learn
 - PA 2
 - Design Questions 2
 - We have gone over all the things you need to complete PA2
- Assignment 1 resubmission is also due next Thursday (1/29)
 - More details on resubmission policy: <https://ucsd-cse29.github.io/wi26/#assignments>



Review Questions

Answers in speaker notes!

For the string “Jéan” answer the following questions:

Q1: What is the UTF-8 encoding?

Q2: What is the UTF-32 encoding?

Q3: What is `strlen` of the UTF-8 encoding? The UTF-32?

J = 74 = 0x4A

é = 233 = 0xE9

a = 97 = 0x61

n = 110 = 0x6E

Answer 1: 0x4A 0xC3 0xA9 0x6E

`strlen` returns 5

Answer 2: 0x00 0x00 0x00 0x4A 0x00 0x00 0x00 0xE9 0x00 0x00 0x00 0x61
0x00 0x00 0x00 0x6E

`strlen` returns 0 for this version (`strlen` is NOT compatible with UTF-32 encoding!)

Another version of UTF-32 depending on byte ordering: 0x4A 0x00 0x00 0x00 0xE9
0x00 0x00 0x00 0x61 0x00 0x00 0x00 0x6E 0x00 0x00 0x00

<https://en.wikipedia.org/wiki/UTF-32>

`strlen` returns 1 for this version

Pointers

string.h

`int strlen(char *s)` class notation would be `int strlen(char s[])`

`void strcpy(char *dest, char *src)` Copies from src to dest

`void strcat(char *dest, char* src)` Appends src to end of dest

`char str[]` and `char *str` mean the same thing in function arguments

`char*`: an address in memory where we can access `char` data

`T*`: an address in memory where we can access T data

T could be `int`, `double`, `int16_t`, `char*` etc

Pointer Types

address = number representing a place in memory (storage in our computer)

For a pointer T^*p :

$p[index]$ looks up the value offset by $index$ from p in memory

$p[index] = v$ change the value offset by $index$ from p to the value v

If x is a variable of type T

$\&x \Rightarrow$ evaluates to a T^* that is the **address** where x is stored

Questions

- Could you return pointers?
 - Yes! This is one of the things that differentiates `char[]` and `char*`.
- Are pointers also stored at an address?
 - Yes, pointers themselves also live somewhere in memory so they have addresses.

Look through this code, what observations do you have?

```
1 #include <stdio.h>
2 #include <stdint.h>
3
4 void wheresmystuff(char* s) {
5     int x = 12;
6     uint8_t y = 53;
7     char z = 9;
8     double ns[] = { 4.0, 3.0, 9.0 };
9
10    printf("x=%d, %lu bytes, starts at: %p\n", x, sizeof(x), &x);
11    printf("y=%hhu, %lu bytes, starts at: %p\n", y, sizeof(y), &y);
12    printf("z=%hhhd, %lu bytes, starts at: %p\n", z, sizeof(z), &z);
13    printf("ns=[%f,%f,%f], %lu bytes, starts at: %p\n",
14          ns[0], ns[1], ns[2], sizeof(ns), &ns);
15
16    printf("s=\"%s\"@%p, %lu bytes, starts at: %p\n", s, s,
17          sizeof(s), &s);
18
19    int main() {
20        char str[] = "14 char string";
21        wheresmystuff(str);
22        printf("\nstr takes up %lu bytes starting at: %p\n",
23              sizeof(str), &str);
24    }
```

Output

```
$ # NOTE - Joe ran this on his Macbook to get this output
$ gcc wheresmystuff.c -o wheresmystuff
$ ./wheresmystuff
x=12, 4 bytes, starts at: 0x16f4c6e44
y=53, 1 bytes, starts at: 0x16f4c6e43
z=9, 1 bytes, starts at: 0x16f4c6e42
ns=[4.000000,3.000000,9.000000], 24 bytes, starts at:
0x16f4c6e50
s="14 char string"@0x16f4c6e98, 8 bytes, starts at:
0x16f4c6e48
str takes up 15 bytes starting at: 0x16f4c6e98
```

`%p` to print pointers

`%f` to print floating-point number
(contrast from `%d`)

Observations

- In the print at line 16, it's interesting that `s` and `&s` are printing different things for the format specifier `%p`
 - `%p` prints out pointers, `&s` is the address of `s`
- `x,y,z` seem to be stored close together (based on the address)
 - Yeah, it's interesting how `x` is at `0x16f4c6e44`, `y` is at `0x16f4c6e43`, and `z` is at `0x16f4c6e42`
- `sizeof` operator returns the size of the entire array and not the pointer for `ns`
 - There are three 8-byte doubles = 24 bytes
- `s` is a 15 character array, but its size is 8 bytes. Why is it not 15?
 - This is because in the context of the function, `s` is a pointer and not an array.

Where are the variables stored?

	Variable/Role	Address	Data							
			0x8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
1	#include <stdio.h>	0x...00								
2	#include <stdint.h>	0x...08								
3		0x...10								
4	void wheresmystuff(char* s) {	0x...18								
5	int x = 12;	0x...20								
6	uint8_t y = 53;	0x...28								
7	char z = 9;	0x...30								
8	double ns[] = { 4.0, 3.0, 9.0 };	0x...38								
9		0x...38								
10	printf("x=%d, %lu bytes, starts at: %p\n", x, sizeof(x), &x);	0x...38								
11	printf("y=%hhhu, %lu bytes, starts at: %p\n", y, sizeof(y), &y);	0x...40								
12	printf("z=%hhhd, %lu bytes, starts at: %p\n", z, sizeof(z), &z);	0x...48								
13	printf("ns={%f,%f,%f}, %lu bytes, starts at: %p\n",	0x...50								
14	ns[0], ns[1], ns[2], sizeof(ns), &ns);	0x...58								
15	printf("s=\"%s\"%02p, %lu bytes, starts at: %p\n", s, s,	0x...60								
16	sizeof(s), &s);	0x...68								
17	}	0x...78								
18		0x...78								
19	int main() {	0x...80								
20	char str[] = "14 char string";	0x...88								
21	wheresmystuff(str);	0x...90								
22	printf("\nstr takes up %lu bytes starting at: %p\n",	0x...98								
23	sizeof(str), &str);	0x...A0								
		0x...A8								
		0x...B0								
		0x...B8								
		0x...C0								
		0x...C8								
		0x...D0								
		0x...D8								
		0x...E0								
		0x...E8								
		0x...F0								
		0x...F8								

```

$ # NOTE - Joe ran this on his Macbook to get this output
$ gcc wheresmystuff.c -o wheresmystuff
$ ./wheresmystuff
x=12, 4 bytes, starts at: 0x16f4c6e44
y=53, 1 bytes, starts at: 0x16f4c6e43
z=9, 1 bytes, starts at: 0x16f4c6e42
ns={4.000000,3.000000,9.000000}, 24 bytes, starts at:
  0x16f4c6e50
s="14 char string"0x16f4c6e98, 8 bytes, starts at:
  0x16f4c6e48
str takes up 15 bytes starting at: 0x16f4c6e98

```

Where are the variables stored? (Answer)

```

1 #include <stdio.h>
2 #include <stdint.h>
3
4 void wheresystuff(char *s) {
5     int x = 12;
6     uint8_t y = 63;
7     char z = 9;
8     double m[] = { 4.0, 3.0, 9.0 };
9
10    printf("x=%d, %i bytes, starts at: %p\n", x, sizeof(x), &x);
11    printf("y=%dhu, %i bytes, starts at: %p\n", y, sizeof(y), &y);
12    printf("z=%dhd, %i bytes, starts at: %p\n", z, sizeof(z), &z);
13    printf("m=%f,%f,%f, %i bytes, starts at: %p\n",
14           m[0], m[1], m[2], sizeof(m));
15
16    printf("a=%\n%Op, %i bytes, starts at: %p\n", s, s,
17           sizeof(s), &s);
18 }
19
20 int main() {
21     char str[] = "14 char string";
22     wheresystuff(str);
23     printf("str takes up %i bytes starting at: %p\n",
24           sizeof(str), &str);
25 }

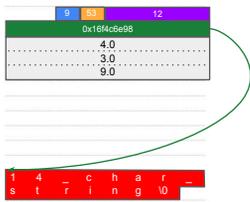
```

```

$ # NOTE - Joe ran this on his MacBook to get this output
$ gcc wheresystuff.c -o wheresystuff
$ ./wheresystuff
x=12, 4 bytes, starts at: 0x16f4c644
y=63, 1 bytes, starts at: 0x16f4c643
z=9, 1 bytes, starts at: 0x16f4c642
m=4.000000,3.000000,9.000000, 24 bytes, starts at:
0x16f4c640
s="14 char string"0x16f4c600, 8 bytes, starts at:
0x16f4c648
str takes up 16 bytes starting at: 0x16f4c600

```

Variable/Role	Address	Data
	0x...00	
	0x...08	
	0x...10	
	0x...18	
	0x...20	
	0x...28	
	0x...30	
	0x...38	
Z, y, X	0x...40	9 63 12
S	0x...48	0x16f4c648
	0x...58	4.0
NS	0x...58	3.0
	0x...68	9.0
	0x...70	
	0x...78	
	0x...80	
	0x...88	
	0x...90	
str	0x...98	1 4 _ c h a r _
	0x...A0	s t r i n g \0
	0x...A8	
	0x...B0	
	0x...B8	
	0x...C0	
	0x...C8	
	0x...D0	
	0x...D8	
	0x...E0	
	0x...E8	
	0x...F0	
	0x...F8	



Questions

- How does the program decide where to put variables?
 - There is an algorithm in gcc to decide this. It would be different on different versions of gcc and computers
- How does sizeof know the size?
 - `sizeof` is a compile-time operator, it means that the compiler calculates the value while it is building your program, not while the program is actually running. As it compiles, a size is determined for every variable

Hashing

Examples of generated hashes

```
$ git commit -m "almost done"
```

```
[main 211935b] almost done
```

```
1 file changed...
```

```
$ git log
```

```
Commit 211935b0cf003 ... (40 hex char = 20 bytes)
```

```
$ ssh jpolitz@ieng6
```

```
ED25519 key fingerprint SHA256: 8avDdt0
```

You have seen
this in lab or doing
the PA!

How are these numbers generated?

`SHA256(char *input, int len char* hash)`

More info in speaker notes 

- `input` is not a C-string and doesn't need to be null terminated
- `len` is the number of chars to process with the hashing function
- `hash` is an output parameter that is written over

Calculates a fixed-size (32 byte) “hash” of that input data.

- deterministic (NOT random)
- “one-way”: hard to guess the input from output
- unpredictably distributed: similar inputs produce very different outputs

`char *input` is JUST a byte array, not a C string. Not every `char*` you see will be a C string.

Joe's Notes (11am)

For this string: `J é a n` (é is code point 233) A1

Q1: What is the UTF8 encoding?

Q2: What is the UTF32 encoding?

Q3: What is strlen of the UTF8 encoding? The UTF-32?

J é a n
 74 233 97 110
 code points: 0x4A 0xEA 0x61 0x6E
 in hex: 0100101010
 0111101001
 UTF32 encoding: 0x00 0x00 0x00 0x4A 0x00 0x00 0x00 0x00 0xEA 0x00 0x00 0x00 0x61 0x00 0x00 0x00 0x6E
 UTF8: 0x4A 0xC3 0xA9 0x61 0x6E
 strlen returns 0 for top version, 1 for bottom version
 strlen returns 5

String.h

`int strlen(char* s)` | `int strlen(char s[])`
 void strcpy(char* dest, char* src) copies bytes from src to dest
 void strcat(char* dest, char* src) appends src to the end of dest

char str[] vs. char* str
 In function arguments, these mean the same thing

char*: an address in memory where we can access char data

T*: an address in memory where we can access T data

T could be

in double
 int64_t
 char*

Pointer Types

address = number representing a place in memory (storage in our computer)

For a pointer T*

p[index] looks up the value offset by index from p in memory

p[index] = v change the value offset by index from p to the value v

x is a variable of type T
 &x evaluates to a T* that is the address where x is stored

Hashing

A3

\$ git commit -m "almost done"
 [main: 211935b] almost done
 1 file changed...

\$ git log
 commit: 211935b0c003... (40 hex char = 20 bytes)

Open File

\$ ss jpolitz@icng6

ED25519 key fingerprint SHA256: Bw-DJ-t0...

SHA256(char* input, int size, char* hash)
 calculates a fixed-size "hash" of that data

- deterministic
- "one-way": hard to guess the input from output
- unpredictably distributed: similar inputs produce different outputs

Joe's Notes (12:30pm)

```

string h | int strlen(char str) A1
int strlen(char str)
void strcpy(char dest, char src) copies src to dest
void strcat(char dest, char src) appends src to end of dest
    
```

~~char s~~ vs. char s[]
 These mean the same thing as function arguments

char* is an address where we can access char data
 T* is an address where we can access T data
 T could be int, float, int, unsigned, char
 T* is a Pointer Type
 pointer = address (of data in memory)
 a number

T* ptr;
 Operations:
 ptr[index] look up data in memory starting at ptr and adding offset of index
 ptr[index] = v change data in memory starting at ptr and adding offset of index

x is variable of type T
 &x evaluates to a pointer (type T*) holding the address where x is stored

Hashing

A3

```

$ git commit -m "almost there!"
[main 0b1a564] almost there!
$ file changed
$ git log
0b1a564a1990... [40 hex chars]
Author: Joe...
    
```

```

$ ssh jpolite@icmg6
The authenticity of host mg6 cannot be established...
... key fingerprint is SHA256:0baDde0... (40 char)
    
```

can make any length
 but a C string of char in input
 ↓ the # of char in input
 void SHA256(char input, int len, char hash)
 ↑ 32 bytes (output param)
 - deterministic - same input produces same hash
 - one-way - cannot compute the input given hash
 - unpredictably distributed - similar inputs have very different hashes