%p - pointer
%f - floating point

AH

```
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=%hhd, %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,
         sizeof(s), &s);
17 }
18
19 int main() {
20   char str[] = "14 char string";
21   wheresmystuff(str);
22   printf("\nstr takes up %lu bytes starting at: %p\n",
         sizeof(str), &str);
23 }
```

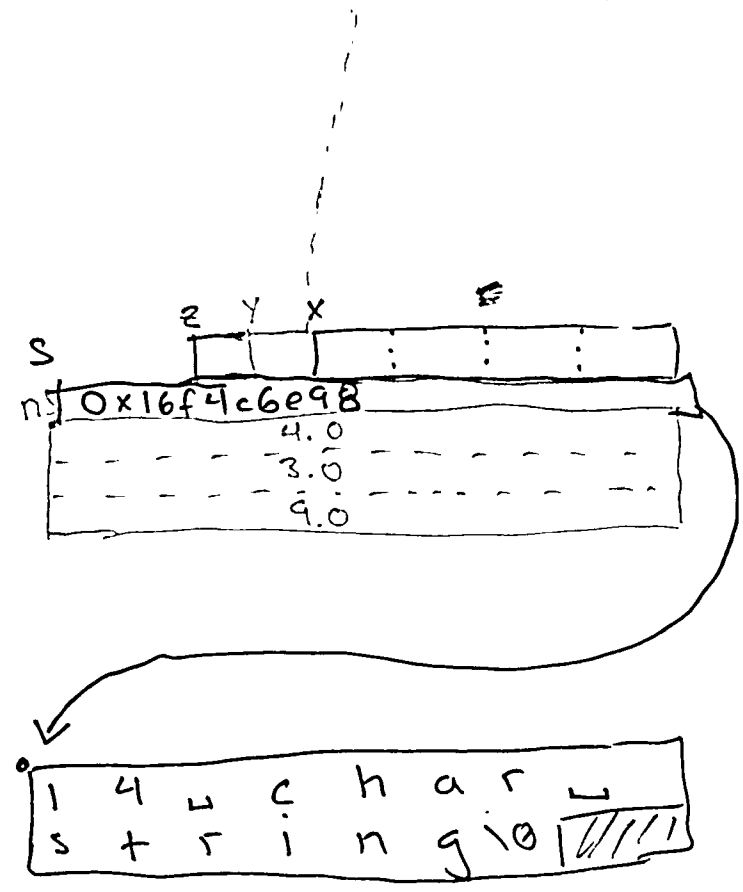
```
$ # NOTE - Joe ran this on his Macbook to get this output
$ gcc wheresmystuff.c -o wheresmystuff
$ ./wheresmystuff
x=12, 4 bytes, starts at: 0x16f4c6e44
y=53, 1 bytes, starts at: 0x16f4c6e43
z=9, 1 bytes, starts at: 0x16f4c6e42
ns=[4.000000,3.000000,9.000000], 24 bytes, starts at:
    0x16f4c6e50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

str takes up 15 bytes starting at: 0x16f4c6e98
```

z y x
s
ns

str

Variable/Role	Address	Data 0/8 1/9 2/A 3/B 4/C 5/D 6/E 7/F
	0x...00	
	0x...08	
	0x...10	
	0x...18	
	0x...20	
	0x...28	
	0x...30	
	0x...38	
	0x...40	
	0x...48	
	0x...50	
	0x...58	
	0x...60	
	0x...68	
	0x...70	
	0x...78	
	0x...80	
	0x...88	
	0x...90	
	0x...98	
	0x...A0	
	0x...A8	
	0x...B0	
	0x...B8	
	0x...C0	
	0x...C8	
	0x...D0	
	0x...D8	
	0x...E0	
	0x...E8	
	0x...F0	
	0x...F8	



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

// vector_sum: takes two vectors represented as double[]
// adds them together component-wise in a new array
// vector_sum({ 1.2, 3.4 }, {-1.0, 3.6 }) => { 0.2, 7.0 }
// Assume the vectors have the same length

// Q: What happens if double[] is used as a return type?
// double[] vector_sum(double vec1[], double vec2[]);

// Q: What about using double* as return type?
// double* vector_sum(double vec1[], double vec2[])

// Pass in length as an argument. Maybe now we've got it!
double* vector_sum(double vec1[], double vec2[], int len) {
  double res[len];
  printf("vec1:\t%p\tvec2:\t%p\tres:\t%p\n", vec1, vec2, res);
  for(int i = 0; i < len; i += 1) {
    res[i] = vec1[i] + vec2[i];
  }
  return res;
}

int main() {
  double v1[] = { 1.3, 4.2 };
  double v2[] = { 1.5, -1 };
  double* v3 = vector_sum(v1, v2, 2);

  double v4[] = { 100, 100 };
  double* v5 = vector_sum(v4, v3, 2);

  printf("first element of v3, v5: %f %f\n", v3[0], v5[0]);

  printf("v1: %p\n", v1);
  printf("v2: %p\n", v2);
  printf("v3: %p\n", v3);
  printf("v4: %p\n", v4);
  printf("v5: %p\n", v5);
}
```

```
$ gcc concat.c -o concat
concat.c:22:10: warning: address of stack memory associated with local
    variable 'result' returned [-Wreturn-stack-address]
   22 |   return result;
      |          ^------
$ ./concat
vec1:   0x16f563010     vec2:   0x16f563000     result: 0x16f562f40
vec1:   0x16f562ff0     vec2:   0x16f562f40     result: 0x16f562f40
first element of v3, v5: 100.000000 100.000000
v1: 0x16f563010
v2: 0x16f563000
v3: 0x16f562f40
v4: 0x16f562ff0
v5: 0x16f562f40
```

Variable/Role	Address	Data
	0x...00	
	0x...08	
	0x...10	
	0x...18	
	0x...20	
	0x...28	
	0x...30	
	0x...38	
	0x...40	
	0x...48	
	0x...50	
	0x...58	
	0x...60	
	0x...68	
	0x...70	
	0x...78	
	0x...80	
	0x...88	
	0x...90	
	0x...98	
	0x...A0	
	0x...A8	
	0x...B0	
	0x...B8	
	0x...C0	
	0x...C8	
	0x...D0	
	0x...D8	
	0x...E0	
	0x...E8	
	0x...F0	
	0x...F8	

Hashing

```
$ git commit -m "almost there!"
[main 0b1a569] almost there!
 1 file changed
```

commit hash

```
$ git log
0b1a569a1490... [40 hex chars]
Author: Joe...
```

```
$ ssh jpolitz@ieng6
The authenticity of host ieng6 cannot be established...
... key fingerprint is SHA256: 8vaDdt0... (~40 char)
```

hash of a "key" on server

can either be any length
not a C string ↓
the # of chars 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

String.h

int strlen(char str[])

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*~~

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
int16_t, double, float, int, uint64_t, 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

%p - pointer
%f - floaty point

```
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=%hhd, %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,
         sizeof(s), &s);
17 }
18
19 int main() {
20   char str[] = "14 char string";
21   wheresmystuff(str);
22   printf("\nstr takes up %lu bytes starting at: %p\n",
         sizeof(str), &str);
23 }
```

```
$ # 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
```

Variable/Role	Address	Data (0/8 1/9 2/A 3/B 4/C 5/D 6/E 7/F)
	0x...00	
	0x...08	
	0x...10	
	0x...18	
	0x...20	
	0x...28	
	0x...30	
	0x...38	
z y x	0x...40	9 53 ← 12 →
s	0x...48	0x16f4c6e98
ns	0x...50	← 4.0 →
	0x...58	← 3.0 →
	0x...60	← 9.0 →
	0x...68	
	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	

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

// vector_sum: takes two vectors represented as double[]
// adds them together component-wise in a new array
// vector_sum({ 1.2, 3.4 }, {-1.0, 3.6 }) => { 0.2, 7.0 }
// Assume the vectors have the same length

// Q: What happens if double[] is used as a return type?
// double[] vector_sum(double vec1[], double vec2[]);

// Q: What about using double* as return type?
// double* vector_sum(double vec1[], double vec2[])

// Pass in length as an argument. Maybe now we've got it!
double* vector_sum(double vec1[], double vec2[], int len) {
  double res[len];
  printf("vec1:\t%p\tvec2:\t%p\tres:\t%p\n", vec1, vec2, res);
  for(int i = 0; i < len; i += 1) {
    res[i] = vec1[i] + vec2[i];
  }
  return res;
}

int main() {
  double v1[] = { 1.3, 4.2 };
  double v2[] = { 1.5, -1 };
  double* v3 = vector_sum(v1, v2, 2);

  double v4[] = { 100, 100 };
  double* v5 = vector_sum(v4, v3, 2);

  printf("first element of v3, v5: %f %f\n", v3[0], v5[0]);

  printf("v1: %p\n", v1);
  printf("v2: %p\n", v2);
  printf("v3: %p\n", v3);
  printf("v4: %p\n", v4);
  printf("v5: %p\n", v5);
}
```

```
$ gcc concat.c -o concat
concat.c:22:10: warning: address of stack memory associated with local
    variable 'result' returned [-Wreturn-stack-address]
   22 |   return result;
      |          ^------
$ ./concat
vec1:   0x16f563010     vec2:   0x16f563000     result: 0x16f562f40
vec1:   0x16f562ff0     vec2:   0x16f562f40     result: 0x16f562f40
first element of v3, v5: 100.000000 100.000000
v1: 0x16f563010
v2: 0x16f563000
v3: 0x16f562f40
v4: 0x16f562ff0
v5: 0x16f562f40
```

Variable/Role	Address	Data
	0x...00	
	0x...08	
	0x...10	
	0x...18	
	0x...20	
	0x...28	
	0x...30	
	0x...38	
	0x...40	
	0x...48	
	0x...50	
	0x...58	
	0x...60	
	0x...68	
	0x...70	
	0x...78	
	0x...80	
	0x...88	
	0x...90	
	0x...98	
	0x...A0	
	0x...A8	
	0x...B0	
	0x...B8	
	0x...C0	
	0x...C8	
	0x...D0	
	0x...D8	
	0x...E0	
	0x...E8	
	0x...F0	
	0x...F8	