

CSc 352

C Programming
Pointers and Arrays

Benjamin Dicken

What will print?

```
char char_var = '\0';  
int int_var = 0;  
long long_var = 0;  
char * char_ptr = &char_var;  
int * int_ptr = &int_var;  
long * long_ptr = &long_var;
```

```
long one, two;
```

```
one = (long) char_ptr; two = (long) (char_ptr+1);  
printf("two minus one: %ld\n", two - one);
```

```
one = (long) int_ptr; two = (long) (int_ptr+1);  
printf("two minus one: %ld\n", two - one);
```

```
one = (long) int_ptr; two = (long) (long_ptr+1);  
printf("two minus one: %ld\n", two - one);
```

```
int values[10] = {0, 50, 100, 150, 200,
                 250, 300, 350, 400, 450};

int i = 0;
while (i < 5) {
    int a = *(values+i) + *(values+9-i);
    printf("%d ", a);
    i += 1;
}
printf("\n");
```

What will print?

The args array

```
int main(●) {  
    . . .  
}
```

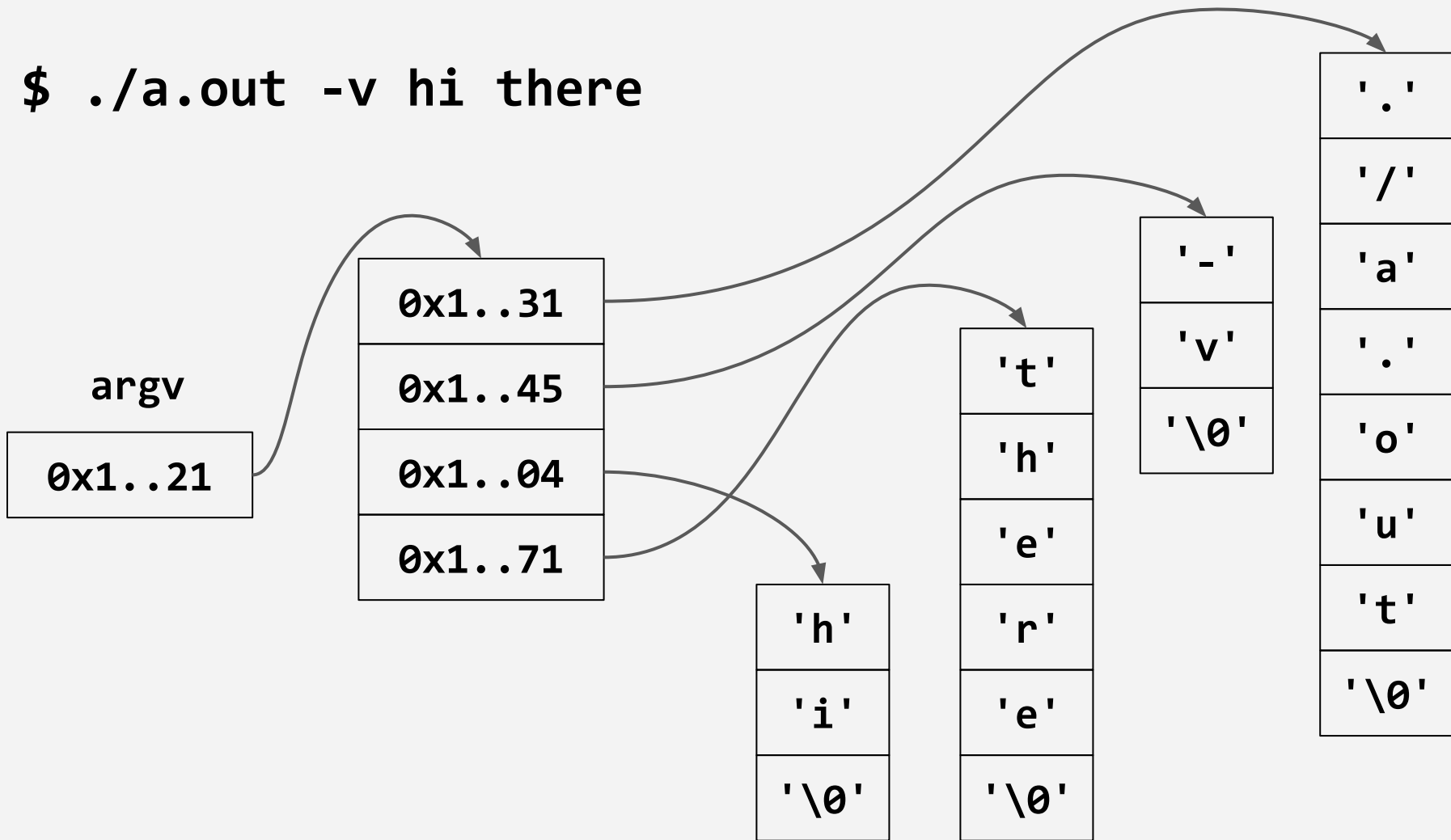


```
int main(int argc, char * argv[]) {  
    . . .  
}
```



This can be thought of as:
“A pointer to an array of
pointers, each of which points
to a sequence of characters”

\$./a.out -v hi there



```
void another (int array[]) {  
    long size = sizeof(array);  
    printf("size: %ld\n", size);  
}
```

```
int main() {  
    int values[10] = {0, 50, 100, 150, 200,  
                     250, 300, 350, 400, 450};  
    long size = sizeof(values);  
    printf("size: %ld\n", size);  
    another(values);  
    return 0;  
}
```

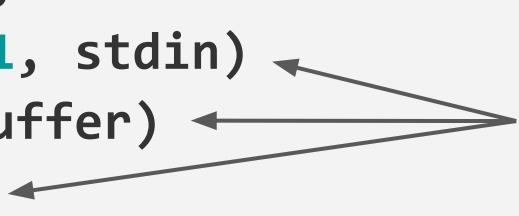
What will happen?
(without -Wall -Werror)

scanf and fgets with pointers

Remember lines such as these?

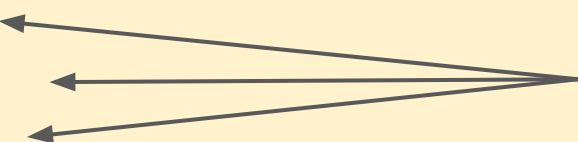
```
int x = 0;  
char buffer[32];  
fgets(buffer, 31, stdin)  
scanf("%31s", buffer)  
scanf("%d", &x)
```

Providing the fgets / scanf functions a *pointer* to the memory location to store the input value

The diagram consists of three arrows originating from a single point on the right side of the text block and pointing to the left. The top arrow points to the 'buffer' argument in the 'fgets' function call. The middle arrow points to the 'buffer' argument in the first 'scanf' function call. The bottom arrow points to the '&x' argument in the second 'scanf' function call.

What is wrong with this code?
What might happen?

```
char last[10];  
char middle[10];  
char first[10];  
printf("first name:\n");  
scanf("%s", first);  
printf("middle name:\n");  
scanf("%s", middle);  
printf("last name:\n");  
scanf("%s", last);  
printf("%s %s %s", first, middle, last);
```



Assume that these three arrays are allocated in sequence in space on the stack (10 bytes, then the next 10, then the next 10)

L-values and R-values

- An L-value is a ***location*** and can be used on the left-hand side of an equals sign
 - Arithmetic-type variables, array elements, pointers
 - Also structs (later)
- An R-value is something that does not actually have a stored location in memory
 - Return value from function, a math expression, etc

Plus Plus

X++ yields x and increments x sometime before or at the completion of the statement it is within.

++X yields $(x+1)$ and increments x sometime before or at the completion of the statement it is within.

```
int x = 1;
int y = 2;
int r1 = 0;
int r2 = 0;
```

What is valid and
what is not valid?

```
r1 = x++;           // A
r2 = (x++)++;      // B
x + y = x + y;     // C
*(&x) = (++y) + (r1++); // D
*(&x) = (++y) + (x++); // E
*(&x + y) = 10;     // F
x++ = y++;         // G
```