Week 3 Friday Array & Strings

Friday, April 16, 2021 4:10 PM

What is An Array?

- collection of elements of the same type
- 1D → Vector 2D → Matrix (array of vectors)
- can have even more dimensions → tensor
- starts at a[0] .... a[n-1]

Declaring An Array

```
type name [count] = {initialization list}
```

- if you have an intilization list, you don't need count
- if you have count, you don't need initialization list
  - but you can have one
  - if list is too short, C will fill rest with zero

Matrices

```
type m[3][3];
```

- elements are [0][0] to [3][3]
- stores in row major order
  - one row in memory, then next, and next
  - memory is one dimensional
  - rows are layed out sequentially

Pointers and Array

- arrays are exception to the rule that parameters are always passed by value
- arrays can be large so you don't want to copy them onto the stack
- the name of the array is pointer to element zero of an array

& (address of) operator

- & gives the memory location (address of a variable)

sizeof operator

- tells us the number of bytes used by a variable
- works for arrays, structures, unions, when the compiler can know how much memory is used

C doesn't know the size of the array because pointers and arrays are the same thing.

```
char *s = "Hello world";
char t[] = "Goodbye cruel world";
```

sizeof(s) = 8 sizeof(*s) = 1
sizeof(t) = 20 sizeof(*t) = 1

sizeof when applied to an array gives the size of the array
sizeof when applied to a pointer gives the size of the pointer

Arrays and pointers in **C** are related in fundamental and often confusing ways.

We will talk more about this when we deal with pointers.

For a single dimensional array,
`a[i] == *(a + i)`

a is the address of `a[0]`	a[i] is the array slot that is at `a + i * sizeof(a[0])`	Pointers automatically do the multiplication by `sizeof`

Pointers and arrays

16 April 2021 © 2020 Darrell Long 13

Slide 16 from Arrays and Strings

Strings

- an array of characters that ends in '\0' (null zero character)
- can be written as
  - char s[] = "Hello World!"
  - char *s = "Goodbye cruel world!"
  - char s[] = {'L', 'c', 'g', 'a', 'l', 0}
- a 100 character (99 printable) empty string is char s[100]

Assignment and Copying

```
char *s = "Wally Wonder Badger";
char *t;

t = s;
```

* taken from slides

- s and t point to (reference) the same memory location
- doesn't make a copy of s