- (Do redirection first)

Structs, PT II

Lecture Overview

- Quick Review
- Before We Code
  - Passing an Array of Structs to a Function
    - Using a pointer
  - Reading from a File into a Struct
  - Opening a File in a Function Other than main
- Sample Programs

QUICK REVIEW

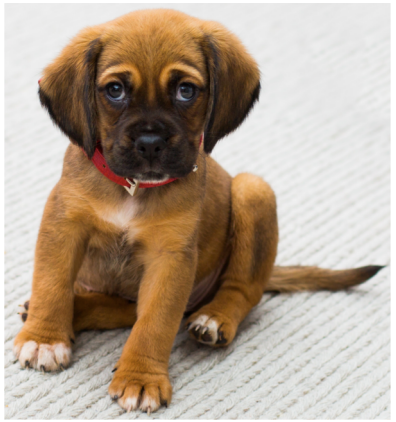
What is a Struct?

```
#include <stdio.h>
#include <string.h>

int main(void)
{
        /*puppy one*/
        char color[]="brown";
        int age_months=4;
        char  breed[]="boxer";

        /*puppy two*/
        char color1[]="white";
        int age_months1=6;
        char  breed1[]="unknown";

}
```



What is a Struct?

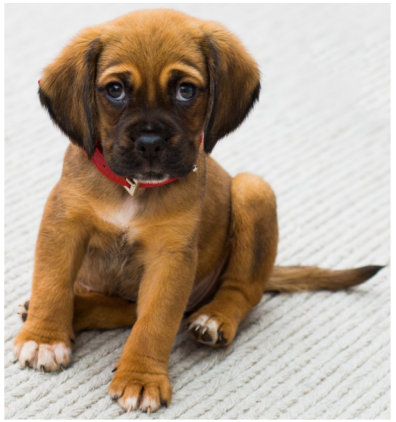
```
#include <stdio.h>
#include <string.h>

struct puppy{

        char color[20];
        int age_months;
        char  breed[20];
};

int main(int argc, char**argv)
{
        /*puppy one*/
        struct puppy p1={"brown", 4, "boxer"};

        /*puppy two*/
        struct puppy p2={"white", 6, "unknown"};
}
```



BEFORE WE CODE

Before We Code

- Today we will be passing an array of structs into a function
- Before we begin, I will revisit the idea of an array of ints
  - Remember we can do pointer arithmetic to move through
- One thing to keep in mind: a lot of beginning programmers in C struggle to keep up with the different ways to write the same code

Remember from a previous lecture:

```
#include "stdio.h"

void example_one(int value[])
{
  printf("Size in function: %lu\n", sizeof(value));
}

void example_two(int* val)
{
  printf("Size in function two: %lu\n", sizeof(val));
}

int main(int argc, char ** argv)
{

   int values[3];
   printf("In main: %lu\n", sizeof(values));

   example_one(values);
   example_two(values);

}
```

Output:

```
computer$ ./a.out
In main: 12
Size in function: 8
Size in function two: 8
```

Notice when passing an array to a function, when using the sizeof operator, we get the size of a pointer…the array is passed as an address, NOT the actual array itself

Using sizeof on the array itself, we get the actual size (12 bytes-3 ints)

You even get a warning about this:

```
(base) Computers-MacBook-Air:C computer$ gcc practice.c
practice.c:10:44: warning: sizeof on array function parameter
 will return size of
      'int *' instead of 'int []' [-Wsizeof-array-argument]
   printf("Size in function: %lu\n", sizeof(value));
                                            ^
practice.c:8:22: note: declared here
void example_one(int value[])
                     ^
1 warning generated.
```

Doing pointer arithmetic on an array of ints Remember, every time we increase the pointer it knows to move ahead by the size of an int (4 bytes on my computer)

```
#include <stdio.h>

void foo(int *n)
{
    printf("%d\n", *n);
    n++;
    printf("%d\n", *n);
}
```

We talked in a previous class about a pointer to an array.

On the next slide, I show you additional ways to do this code

```
int main(int argc, char **argv)
{
    int nums[]={2, 3, 4};
    int *ptr=nums;
    foo(ptr);
}
```

Note we don't need the address operator since nums is treated as an address to the first element

All these programs behave in the same way like the program on the previous slide:

1

```
#include <stdio.h>

void foo(int n[])
{
        printf("%d\n", *n);
        n++;
        printf("%d\n", *n);
}

int main(int argc, char **argv)
{
        int nums[]={2, 3, 4};
        foo(nums);
}
```

2

```
#include <stdio.h>

void foo(int n[])
{
        printf("%d\n", n[0]);
        printf("%d\n", n[1]);
}

int main(int argc, char **argv)
{
        int nums[]={2, 3, 4};
        foo(nums);
}
```

3

```
#include <stdio.h>

void foo(int *n)
{
        printf("%d\n", *n);
        n++;
        printf("%d\n", *n);
}

int main(int argc, char **argv)
{
        int nums[]={2, 3, 4};
        foo(nums);
}
```

Remember an array is the address to the first element (like a pointer)

4

```
#include <stdio.h>

void foo(int *n)
{
        printf("%d\n", n[0]);
        printf("%d\n", n[1]);
}

int main(int argc, char **argv)
{
        int nums[]={2, 3, 4};
        foo(nums);
}
```

Remember from a previous lecture: (use of brackets)

When I code:

```
int arr[]={3, 4, 5, 78, 19, 99}
```

We can think of it like this:

0	**1**	**2**	**3**	**4**	**5**
3	4	5	78	19	99

arr

When I refer to arr[1], I'm referring to this value at index 1

What we really mean is start at the first address of the array and each index refers to number of bytes each element takes (ints take 4 bytes):

x100	x104	x108	x112	x116	x120
- - - -	- - - -	- - - -	- - - -	- - - -	- - - -
3	4	5	78	19	99

So, starting at x100, we have 4 bytes to hold *3*. We refer to this first chunk of bytes as index 0. The second chunk of bytes (x104-x107) holding *4* is referred to as index 1 and so on.

We can't do pointer arithmetic on an array (we don't want to lose the first address), but we can do it to a pointer at the array (so no arr++).

Note we can modify elements in an array (but NOT single elements)

```
#include "stdio.h"

void example_one(int value[])
{
  value[1]=40;
}

int main(int argc, char ** argv)
{
   int values[]={1,2,3};

   example_one(values);
   printf("Changed value: %d\n",values[1]);
}
```

Output:

```
computer$ ./a.out
Changed value: 40
```

We can change the value here because we are passing in the array (by address)

```
#include "stdio.h"

void example_one(int value)
{
  value=40;
}

int main(int argc, char ** argv)
{
   int values[]={1,2,3};

   example_one(values[1]);
   printf("Changed value: %d\n",values[1]);
}
```

Output:

```
computer$ ./a.out
Changed value: 2
```

We can't change the value here because we are passing in the single element (by value)

Before We Code

- It is the same idea with arrays of structs
  - They are arrays, just like the int arrays on the previous slides
  - The pointer moves ahead by whatever size the struct element of the array is (like an int pointer moves ahead 4 bytes because that's the size of an int on my computer)
- On the following slides, I will show you different variations of the same program
  - Passing arrays of structs to functions
  - The numbers at the top correspond to the int array examples on the previous slides

1

```
#include <stdio.h>

typedef struct irish{
        char irish_name[20];
        char gender;
        char english_equiv[20];
}irish;

void foo(irish str[])
{
        (*str).gender='m';
        str++;
        (*str).gender='f’;
}

int main(int argc, char **argv)
{
        irish names[13];
        foo(names);
        printf("%c", names[0].gender);
        printf("%c", names[1].gender);
}
```

2

```
#include <stdio.h>

typedef struct irish{
        char irish_name[20];
        char gender;
        char english_equiv[20];
}irish;

void foo(irish str[])
{
        str[0].gender='m’;
        str[1].gender='f’;
}


int main(int argc, char **argv)
{
        irish names[13];
        foo(names);
        printf("%c", names[0].gender);
        printf("%c", names[1].gender);
}
```

3

```
#include <stdio.h>

typedef struct irish{
        char irish_name[20];
        char gender;
        char english_equiv[20];
}irish;

void foo(irish *ptr)
{
        (*ptr).gender='m';
        ptr++;
        (*ptr).gender='f';
}

int main(int argc, char **argv)
{
        irish names[13];
        foo(names);
        printf("%c", names[0].gender);
        printf("%c", names[1].gender);
}
```

4

```
#include <stdio.h>

typedef struct irish{
        char irish_name[20];
        char gender;
        char english_equiv[20];
}irish;

void foo(irish *ptr)
{
        ptr[0].gender='m';
        ptr[1].gender='f';
}

int main(int argc, char **argv)
{
        irish names[13];
        foo(names);
        printf("%c", names[0].gender);
        printf("%c", names[1].gender);
}
```

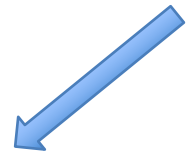
We can modify because we're passing an array in (address)

```
#include <stdio.h>

typedef struct irish{
        char irish_name[20];
        char gender;
        char english_equiv[20];
}irish;

void foo(irish *ptr)
{
        ptr->gender='m';
        ptr++;
        ptr->gender='f';
}

int main(int argc, char **argv)
{
        irish names[13];
        irish *ptr=names;
        foo(ptr);
}
```



You can also use the arrow notation I showed you in the last class

(*ptr).gender is the same as **ptr->gender**

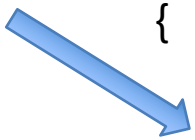
Finally, we could just make a pointer to the array of structs and pass that

```
#include <stdio.h>

typedef struct irish{
        char irish_name[20];
        char gender;
        char english_equiv[20];
}irish;

void foo(irish *ptr)
{
        (*ptr).gender='m';
        ptr++;
        (*ptr).gender='f';
}

int main(int argc, char **argv)
{
        irish names[13];
        irish *ptr=names;
        foo(ptr);
}
```



Note we don't need the address operator since names is an address to the first element

```
void foo(irish str[])
{
        str[0].gender='m';
        str[1].gender='f';
}
```

```
void foo(irish *ptr)
{
        ptr[0].gender='m';
        ptr[1].gender='f';
}
```

When we use a pointer/address and we use the brackets, it's the same idea as the array on the previous slide.

You can use the sizeof operator to see the size of a struct: sizeof(irish). We then know how many bytes the index for the array is covering. *Note-this works where you declared the struct-not if you passed it by address*

```
void foo(irish str[])
{
        (*str).gender='m';
        str++;
        (*str).gender='f';
}
```

```
void foo(irish *ptr)
{
        (*ptr).gender='m';
        ptr++;
        (*ptr).gender='f';
}
```

Even though we can't use ++ on an array, since we are passing the array as an address to the function, we can use ++ to move to the next index (chunk of bytes)

Don't forget you can use ->

Before We Code

- You will see me reading from a file into a struct
  - Use strtok to separate out parts
  - I will use fopen and also show an example of redirection

SAMPLE PROGRAMS

Sample Program 1

- Create a program that makes fortune cookies. Each cookie has a message and six lucky numbers. This information is kept in a file (use **redirection** to read in info).
- Once the cookies are created, output information about a specific cookie
  - To screen
  - To a file (using **redirection**)

Sample Program 2

- Conchobhar wants to create a program that allows users to type in either an Irish name or its English equivalent. The program should then output to screen the corresponding name (so if they type in an Irish name, the program should print out the English version of that name).