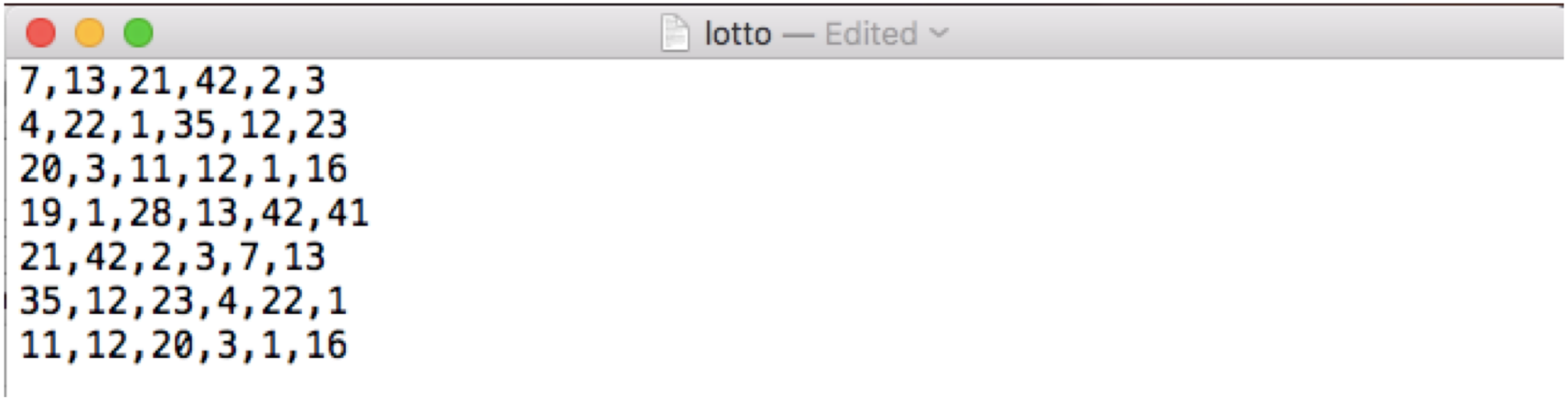
Practice Program

- Create a program that reads in daily lotto numbers (see file below) and prints them out if the user wants to see them



Structs

Lecture Overview

- Lecture
  - Structs
    - What is a Struct?
    - Defining a Struct
    - Array of Structs
- Before We Code
  - Passing a Struct to a function
  - Pointers at Structs
  - Naming the executable program
- Sample Programs

LECTURE

What is a Struct?

Defining a Struct

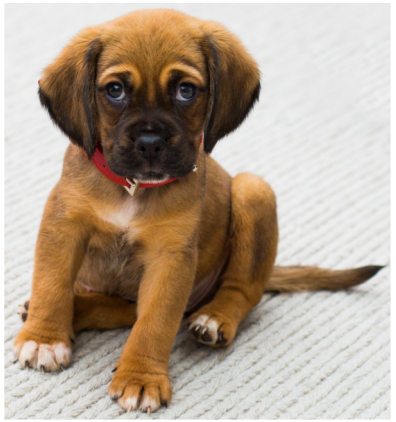
Array of Structs

What is a Struct?

- A struct is a way to make our code reusable and more organized
- We can encapsulate all the characteristics of an object or topic into one structure
  - Instead of having our variables scattered around our program
- You can think of it like an object in Java

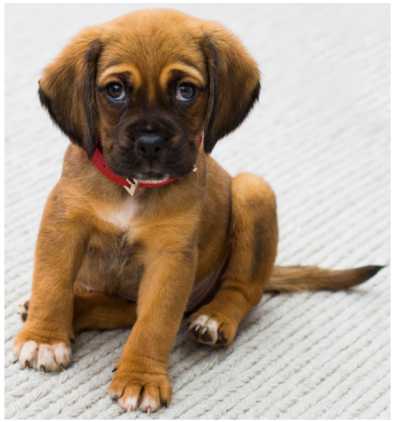
What is a Struct?

Let’s say that we want to describe these puppies in our program. How would we do that?



What is a Struct?

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

int main(int argc, char**argv)
{
        /*puppy one*/
        char color[]="brown";
        int age_months=4;
        char gender='f';
        char  breed[]="boxer";

        /*puppy two*/
        char color1[]="white";
        int age_months1=6;
        char gender1='m';
        char  breed1[]="unknown";

}
```

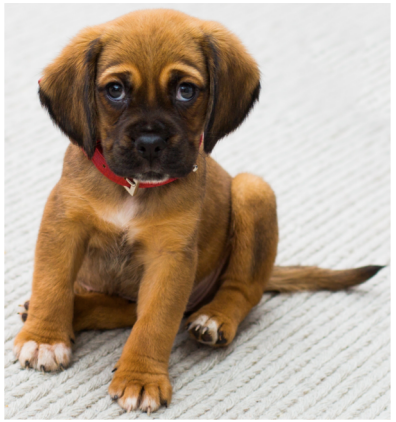


What is a Struct?

- The program shown on the previous slide repeats a lot of code
- How can we make this program better?
  - How can we make our code reusable?
- We can create a puppy struct
  - Every time we want to talk about a puppy in our program, we simply use the puppy struct

What is a Struct?

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

struct puppy{

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

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

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



What is a Struct?

Defining a Struct

Array of Structs

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

struct Candy{
	char name[20];
	int calories;
};

int main(int argc, char**argv)
{
	struct Candy candy1={"Skittles", 180};
	struct Candy candy2={"Starburst", 290};

	FILE *fp=fopen("somefile.txt", "w+");

	if(fp==NULL)
	{
		printf(“fp is NULL”);
	}

	else
	{
		fprintf(fp,"%s,%d", candy1.name, candy1.calories);
	}
}
```

1. We can define it using the word struct

When we want to use our struct, we do this

(note that we are initializing our struct as soon as we create it here)

Notice we are using . to access members of our struct
We can also write to a file with our struct values

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

struct Candy{

        char name[20];
        int calories;

};

typedef struct Candy C;

int main(int argc, char**argv)
{
        C candy1={"Skittles", 180};
        C candy2={"Starburst", 290};
}
```

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

typedef struct Candy{
        char name[20];
        int calories;
} C;

int main(int argc, char **argv)
{
        C candy1={"Skittles", 180};
        C candy2={"Starburst", 290};
}
```

2. We can use the word typedef

This way, we don't have to use the word struct everytime we want to use the struct.

What is a Struct?

Defining a Struct

Array of Structs

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

struct Candy{
        char name[20];
        int calories;
};

int main(int argc, char**argv)
{
        struct Candy candy[2]={{"Skittles", 180}, {"Starburst", 290}};
        FILE *fp=fopen("somefile.txt", "w+");

        if(fp==NULL)
        {
                printf(“fp is NULL”);
        }

        else
        {
                fprintf(fp,"%s,%d", candy[0].name, candy[0].calories);
        }
}
```

Just like with other types that we have dealt with, we can create arrays of structs.

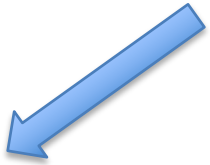
We can access the structs the same way we do with other types. Notice to access the specific member we use . (dot)

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

struct Candy{
	char name[20];
	int calories;
};

int main(int argc, char**argv)
{
	struct Candy candy1={"Skittles", 180};
	struct Candy candy2={"Starburst", 290};

	struct Candy both_candy[2];
	both_candy[0]=candy1;
	both_candy[1]=candy2;
}
```

We can create the individual structs and give the initial values

We create an array of structs and assign the structs we created to the array

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

struct Candy{
        char name[20];
        int calories;
};

int main(int argc, char**argv)
{
        struct Candy candy1;
        struct Candy candy2;
        char line[10];
        int n;

        printf("Enter candy name: ");
        scanf("%s", line);
        strcpy(candy1.name, line);

        printf("Enter candy calories: ");
        scanf("%s", line);
        candy1.calories=atoi(line);

}
```



We can enter information into our structs

BEFORE WE CODE

Before We Code

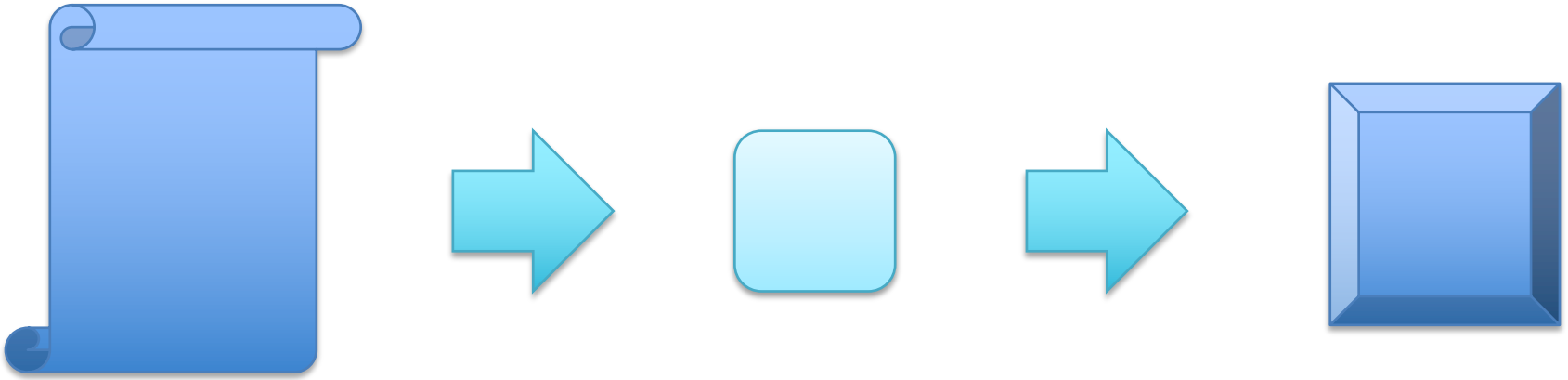
- I will talk a bit about pointers to structs
- You will see me passing structs to functions
  - Passing a struct to a function
  - Passing a struct pointer to a function
    - Using the -> notation

Before We Code

- So far, we have been using ./a.out when we run our program.
- What does this mean exactly and do we always have to use it?

(I may have mentioned this at the beginning of the semester)

Before We Code



Your C Code

This is not directly executable by the computer.

Compiler

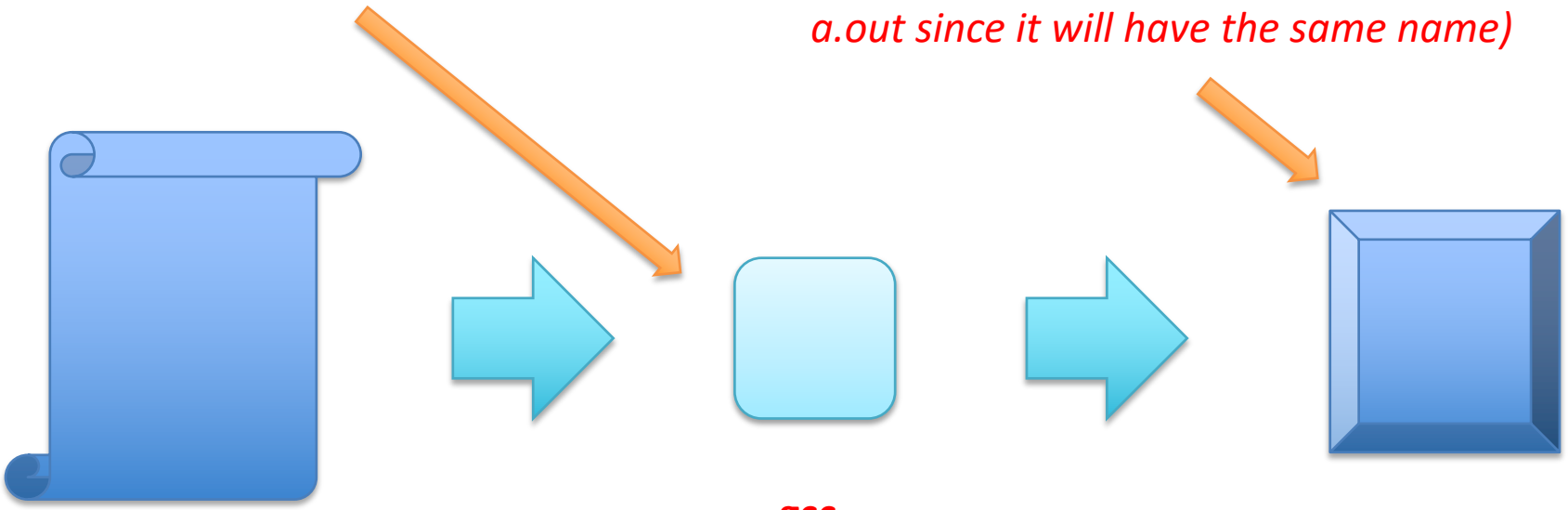
Compile your C code (turn it into something the computer can understand)

Executable Program

Your computer can actually execute this

We use the gcc compiler

The default name of our executable program is a.out. If we do not give a name to our executable it will always be called a.out (and every new executable will overwrite the old a.out since it will have the same name)



Your C Code

This is not directly executable by the computer.

gcc
Compiler

Compile your C code (turn it into something the computer can understand)

a.out
Executable Program

Your computer can actually execute this

Before We Code

Desktop computer$ gcc movie.c
Desktop computer$ ./a.out

This is what we have been doing in class. Notice we use a.out to run our program.

Desktop computer$ gcc -o movieprog movie.c
Desktop computer$ ./movieprog

Here, we name our executable (instead of leaving it as a.out). Notice how we use the name we give to run the program

SAMPLE PROGRAMS

Program 1

- Create a program that allows the user to enter in candy information (the name of the candy and the number of calories). The program should then tell the user which candy has more calories.

Program 2

- Create a program that allows a user to enter in two movies and the ratings for these movies.