CS 2113
Software Engineering

Lecture 4:
C programming - Structures, memory, and files

Professor Tim Wood - The George Washington University

This Time

Review tricky parts of EX3

More on structs, malloc, strings, files

Lots of C practice

Get ready!

Do this now!!!

Open Code Anywhere

```
git clone https://github.com/cs2113f16/lec-4.git
```

Refresh your file view

Structs and Pointers

Worksheet 1

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

struct car {
  char make[50];
  char model[50];
  int year;
};

int main(){
  struct car c1;
  struct car* c2;

  strcpy(c1.make, "Honda");
  strcpy(c1.model, "Fit");
  c1.year = 2015;

  strcpy(c2->make, "Fiat");
  strcpy(c2->model, "500");
  c2->year = 2016;
  return 0;
}
```

Stack

Address	*Name*	*Contents*
10000		

Heap

Address	*Allocated?*	*Contents*
50000		

Structs and Pointers

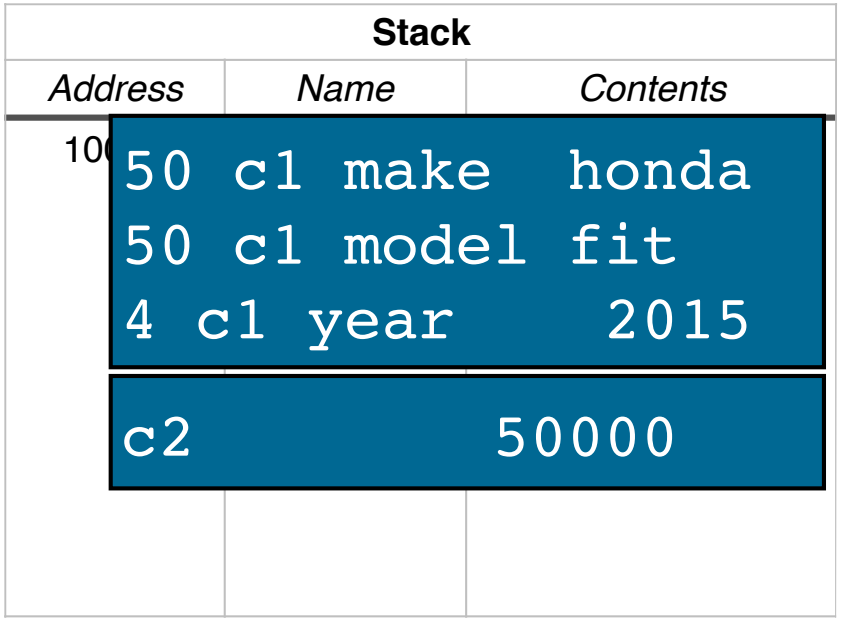
Worksheet 1

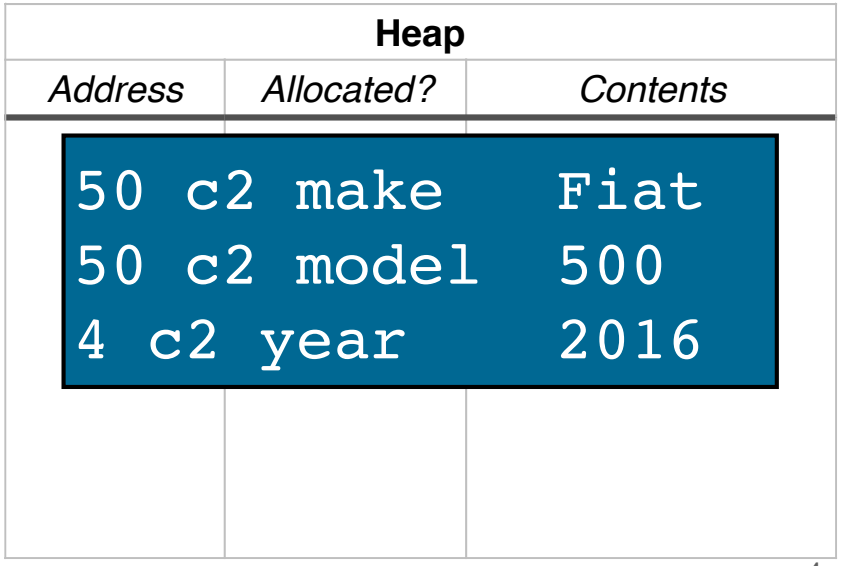
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct car {
  char make[50];
  char model[50];
  int year;
};

int main(){
  struct car c1;
  struct car* c2;
  c2 = malloc(sizeof(struct car));
  strcpy(c1.make, "Honda");
  strcpy(c1.model, "Fit");
  c1.year = 2015;

  strcpy(c2->make, "Fiat");
  strcpy(c2->model, "500");
  c2->year = 2016;
  return 0;
}
```





Structs and Pointers

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

struct car {
  char make[50];
  char model[50];
  int year;
};

int main(){
  struct car c1;
  struct car* c2;

  strcpy(c1.make, "Honda");
  strcpy(c1.model, "Fit");
  c1.year = 2015;

  strcpy(c2->make, "Fiat");
  strcpy(c2->model, "500");
  c2->year = 2016;
  return 0;
}
```

If c2 is a pointer, how come we don't have to use a * when dereferencing it to change the values inside of it?

(*c2).make

c2->make

fgets

Handy way to read from a file or get input from user

```
char *fgets ( char *str, int num, FILE * stream );
```

What does all of this mean?

fgets

Handy way to read from a file or get input from user

```
char *fgets ( char * str, int num, FILE * stream );
```

return type of function. Don't be tricked by placement of the *!

Remember, a char* and a char[] are basically the same!

Typically want to use sizeof(string)

fgets

Handy way to read from a file or get input from user

```
char *fgets ( char * str, int num, FILE * stream );
```

Annoying thing: includes the '\n' in str!

How can we fix this?

fgets

```
char *fgets ( char * str, int num, FILE * stream );

char string[50]; (have space for 50 bytes)
strcpy(string, "abc\n"); (using 5 bytes)
strlen(string) --> 4 letters (4 bytes)

str[strlen(str) -1] = '\0';
```

How can we fix this?

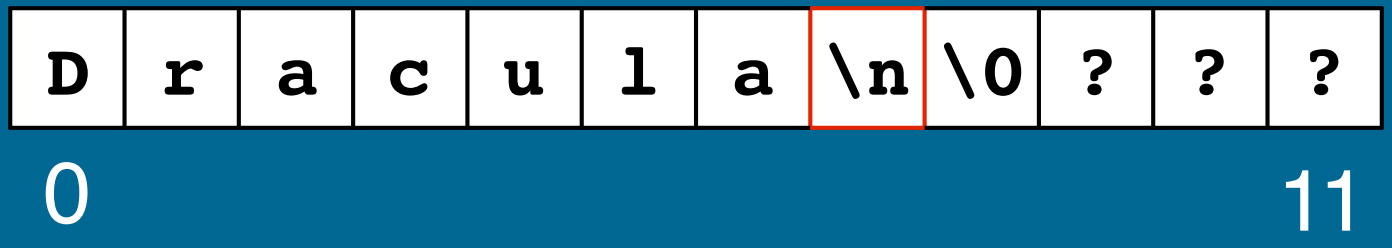
- use strlen() function to find length of the string
- change the last symbol to be ‘\0’ instead of ‘\n’

Worksheet 2

- Write a helper function in **`lec-4/clean_newline.c`**

```
char* clean_newline(char * str);
```

- Return pointer to the same string
- Remove a **\n** character at the end of **str**



- **New:** To compile your code, just run **`make`**
  - To run the code, type **`./cleaner`**

Worksheet 2

- Write a helper function in **`lec-4/clean_newline.c`**

```
char* clean_newline(char * str);
```

- Return pointer to the same string
- Remove a **\n** character at the end of **str**

- **How robust is your solution?**

- Call the **`test_clean_newline()`** function and see if you pass...

Worksheet 2

- Write a helper function in **`lec-4/clean_newline.c`**

```
"hello\n"

str[strlen(str) -1] = '\0';

char * str = "";

char * str2 = "hello";
```

Testing

Our first **Software Engineering principle**!

When you write code, you should think and design **test cases** to evaluate whether your code works correctly

- What are the edge cases?
- Empty variables? Missing files?
- Make your program fail gracefully if there is a problem instead of causing undefined behavior!

Multi-file C

Most programs don't fit into a single C file

You can split your code into multiple files:

```
gcc clean_newline.c tests.c
```

Cannot have multiple functions with the same name

- They are all going to be combined into one executable so names will clash!
- Need to have a **main** function in exactly one file!

Global variables defined in one file are not accessible in another unless you use the **extern** keyword

- More on this another day in another class...

Multi-file C

Header files define function prototypes

- Tells the compiler that it should expect to find a copy of the named function somewhere

```
// tests.h
char* clean_newline(char* str);
void test_clean_newline();
```

- Header files (generally) do not contain the implementation of functions

Need to #include header files--just pastes contents

```
#include <stdio.h>
#include "tests.h"
```

local, user defined library in " "

Make Files

`make` is a tool to help you compile complex projects

- Not C specific! Can work with many languages
- Just automates the task of typing in commands like "javac ..."

```
# Simplest Makefile

hello:
   gcc hello.c -o hello
```

thing you want to build

compilation command

Defines a "build target" named hello

- Runs the gcc command to compile a file

Can have multiple targets and variables defined

More Complex Make

Make can track dependencies

- Checks whether a file has been modified to decide whether it needs to be compiled

```
# Simplest Makefile with dependencies
CFLAGS=-g

hello: hello.c
   gcc hello.c -o hello $(CFLAGS)
```

This code will only compile hello.c if it has been changed more recently than the build target

Even More Complex Make

Can have multiple build targets with linked dependencies

```
CFLAGS=-g
DEPS=tests.h

cleaner: tests.o clean_newline.o
   gcc clean_newline.o tests.o -o cleaner $(CFLAGS)

tests.o: tests.c $(DEPS)
   gcc -c tests.c $(CFLAGS)

clean_newline.o: clean_newline.c $(DEPS)
   gcc -c clean_newline.c $(CFLAGS)
```

Space for strings

This was kind of wasteful

```
struct car {
  char make[50];
  char model[50];
  int year;
};
//…
strcpy(c1.make, "Honda");
strcpy(c1.model, "Fit");
```

How could we use less memory for these strings?

Space for strings

Becomes an even bigger issue when we have lots of strings

Suppose we need to read in 20 lines of Romeo & Juliet so that we can do linguistic analysis…

```
char lines[20][100];

for i=0 to 20:
  fgets(tmpline, 100, rj_file)
  strcpy(lines[l], tmpline);
```

How much space **should** we need?

Worksheet #3

Need to use malloc!

Becomes an even bigger issue when we have lots of strings

Suppose we need to read in 20 lines of Romeo & Juliet so that we can do linguistic analysis…

```
create a  char* lines[20] --> an array of 20 things, each a string
read a line from the file into a temp string
if there was an error, break out of for loop
find out the length of the line that we just read
lines[i] = malloc(strlen +1) --> allocate heap space, including \0
copy the temporary line into lines[i]
```

Need to use malloc!

Becomes an even bigger issue when we have lots of strings

Suppose we need to read in
Juliet so that we can do lingu

Stack

Address	*Name*	*Contents*
10000	lines[0]	50000
10004	lines[1]	50006
10008	lines[2]	50015
	…	
10076	lines[19]	

Heap

Address	*Alloc?*	*Contents*
50000	Y	"Hello\0"
50006	Y	"Hi Romeo\0"
50015	Y	"How are you?\0"

read a line from the file into a te

if there was an error, break out

find the length of the temp string

allocate enough space for the temp string and store the address into the next entry in the array

copy the temporary line into the new space

Think in Algorithms

(Not code!)

When you are given a problem, try to think about it more abstractly first—don't rush to code

- Write on paper what steps your program needs to take

If you can't figure out what to do, break it down into a smaller problem and solve that

Once you have pseudo code of your algorithm, then try to write the real code

- If you can't figure out what to do, break it down into a smaller problem and solve that
  - If you can't figure out what to do, break it down into a smaller problem and solve that
    - If you can't figure out what to do, break it down into a smaller problem and solve that

Ex-4: HaikuGen2000

- Read Haikus from a file:

No keyboard present
Hit F1 to continue
I can only cry.

A crash reduces
Your expensive computer
To a simple stone.

Windows Seven crashed.
I am the blue screen of death.
No one hears your screams.

- All haikus are 3 lines long
  - Line 1 and line 3 have FIVE syllables
  - Line 2 has SEVEN syllables

```
char* shortLines[200];
```

```
char* longLines[100];
```

- You will be given a long file with lots of haikus
- Your goal is to randomly recombine lines from different haikus to produce a new one:

To a simple stone.
Hit F1 to continue
No one hears your screams.