Midterm Notes

Type	test review
Materials	
Reviewed	☐

Bash is an interpreter.

There is no get opts and heredoc.

C is the major chunk.

The seven types of files are:

1. Regular files
2. Directories
3. Symbolic links
4. Named pipes
5. Character devices
6. Block devices
7. Sockets

There is no further information from chapter 4.

Chapter 3 makes up 80% of the test.

The steps from .c to a.out are: (-E) → .p (-S) → .s (-C) → .o (linking) → a.out.

You need to know how to write a makefile.

".o" is used to specify an object name.

The four regions of a program are: #big_num stack → heap → data → text #0.

Function pointers are stored in text

Function pointers are stored in the "text" region of a program because they are essentially just pointers to the code of a function. The "text" region contains the executable code of a program, so it makes sense to store function pointers there.

Notes from book

> chapters 1-4

UNIX Kernel and Kernel API

The UNIX operating system is called the **Kernel**. This is loaded into memory called **system space**

The kernel provides the following services

- I/O handling
- file-management
- multi-tasking
- multi-threading
- device management

to switch from user mode to kernel mode, the processors needs to change its execution mode. this is done by implementing a **trap instruction**

the kernel is small so they use libraries to provide more complicated commandds

Bash scripts

default output destination is stdout (the terminal)

To redirect the location of the output we use **“>”** special character

To append to the existing file we would use “>>”

> 💡 To redirect the output to a file, use the > character followed by the name of the file. For example, "ls > file.txt" will write the output of the "ls" command to a file named "file.txt" instead of printing it to the terminal. Similarly, to append output to an existing file, use the ">>" character. For example, "echo 'hello' >> file.txt" will append the word "hello" to the end of the file "file.txt".

to get input from a file you can use “<”

special character “|” is using to feed the output of one program into the input for another

💡 In Bash scripts, the default input source is stdin (standard input). To redirect the input to come from a file, use the "<" character followed by the name of the file. For example, "sort < file.txt" will sort the contents of "file.txt" instead of using input from the keyboard. Additionally, there are other special characters that can be used for input and output redirection, such as pipes ("|") and process substitution ("<()").

> to make output disappear you can redirect it to /dev/null

C language

there is no native boolean type, use 0 or 1 and there is also no native strings

C strings are null-terminating so when you declear char by char you need 0 at the end

size_t is a unsigned integer used to avoif negitive numbers in loops

Structs are a way to define custom data types in C. They allow you to group together related variables of different types under a single name. This makes it easier to organize and manipulate complex data structures.

you can use typedef to avoid having to put struct infront of the new variablw you want

done like this

```
typedef struct{
  int width;
  int height;
} Rectangle;

//used like this
Rectangle rect1;
// now has all the properties of a Rectangle
rect1.height;
```

all members of the struct are public

Functions

The shell passes the executable file a.out as a char array (6 elements) to the argv array. The first element is the name of the executable file.

Macros

A macro is basically a symbolic representation done by...

```
#define PI 3.14
```

You can pass arguments to them like functions. They can also create conditionals like this:

```
#define Finished
int main(int argc, char** argv) {
  #ifdef DRAFT
      printf("I'm a draft!\\n");
  #else
      printf("I'm finished!\\n");
  #endif
  return 0;
}
```

They are mostly used for error checking and only compiling specific sections of code.

Preprocessor

Anything that starts with a hashtag are directives for the processor. gcc runs the preprocessing.

Exit

`return(0)` returns to the caller while `exit` will terminate the program regardless of where it is. `return` calls back to the C start-up routine which calls `exit` for us. `exit` first calls exit handlers and the standard I/O cleanup routine. Once done, it calls `_exit()`, which is a system call to actually exit the process completely.

Pointers

The address of an operator is denoted by &.

To obtain the value instead of the pointer, we use the * operator in front of a pointer.

When working with arrays and pointers, we need to shift by i + array and then dereference that.

C is a pass-by-value language, meaning that when we pass variables as arguments, only a copy is sent to the function's variables. The function has its own allocated space in memory where anything done in it will stay, and will be destroyed once the function terminates.

> go back to page 36

REVIEW SESSION

> add picture of linex subsystems

> special character and fd used to redirect and stand error >&2

input is 0

output is 1

error is 2

shabang !!!!! is the first line in bash

> GCC with flags

.h has function declaration

> go over previous year coding assignment

`realloc()`

is used to make more space is needed later in program

void pointer

cannot deference

cannot use arrhythmic compression

```
#include <stdio.h>

int square(int num) {
  return num * num;
}

void doMath(int num, int (*function_ptr)(int)){

  int result = function_ptr(num);

  printf ("Result: %d\n",result);
}

int main() {

  int (*func_ptr)(int) = &square;

  doMath(5, func_ptr);
  return 0;
}
```

💡 malloc allocates memory on the heap but the address of memory is on the stack

target

depdencies

commands

compile time errors are more logical while run time is more syntax

seg fault is memory of the heap