This is CS50.
Week 4

- Hexadecimal
- Pointers
- Dynamic Memory
- Memory Layout
- File I/O
What questions do you have?
Questions
Today

Pointers

File I/O

Dynamic Memory
PART ONE

Pointers
Pointers
Types

**int**  Integer

**char**  Character
Types

int * Address of an Integer
char * Address of a Character
Pointers

• `type *` is a pointer that stores the address of a `type`
• `*x` takes a pointer `x` and goes to that address
• `&x` takes `x` and gets its address
int a = 28;
int a = 28;
int a = 28;
int a = 28;
int a = 28;
int a = 28;
int b = 50;
int a = 28;
int b = 50;
int a = 28;
int b = 50;
int a = 28;
int b = 50;
int *c = &a;
int a = 28;
int b = 50;
int *c = &a;
int a = 28;
int b = 50;
int *c = &a;
int a = 28;
int b = 50;
int *c = &a;
*c = 14;
int a = 28;
int b = 50;
int *c = &a;
*c = 14;
int a = 28;
int b = 50;
int *c = &a;
*c = 14;
c = &b;
int a = 28;
int b = 50;
int *c = &a;
*c = 14;
c = &b;
int a = 28;
int b = 50;
int *c = &a;
*c = 14;
c = &b;
*c = 20;
int a = 28;
int b = 50;
int *c = &a;
*c = 14;
c = &b;
*c = 20;
Memory Layout
Stack
Stack

local variables and functions
Heap

Stack

local variables and functions
Heap

- dynamically allocated memory

Stack

- local variables and functions
Stack

Heap

dynamically allocated memory

local variables and functions
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
int main(void)
{
    f();
}

void f(void)
{
    g();
h();
}
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
void swap(int a, int b) {
  int tmp = a;
  a = b;
  b = tmp;
}
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
void swap(int a, int b) {
    int tmp = a;
    a = b;
    b = tmp;
}
void swap(int a, int b) {
    int tmp = a;
    a = b;
    b = tmp;
}
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}

swap(x, y)
void swap(int a, int b) {
    int tmp = a;
    a = b;
    b = tmp;
}
void swap(int a, int b) {
    int tmp = a;
    a = b;
    b = tmp;
}
void swap(int a, int b) {
    int tmp = a;
    a = b;
    b = tmp;
}
void swap(int a, int b) {
    int tmp = a;
    a = b;
    b = tmp;
}
```c
void swap(int a, int b) {
    int tmp = a;
    a = b;
    b = tmp;
}
```
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```c
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```

```c
swap(&x, &y)
```

```
x [50]  y [100]
```
void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

swap(&x, &y)
void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

swap(&x, &y)
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

swap(&x, &y)
void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

swap(&x, &y)
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```c
void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```

void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
PART TWO
File I/O
File I/O
Open a File

```c
FILE *f = fopen("filename.txt", "r");
```
Open a File

```c
FILE *f = fopen("filename.txt", "r");
```

Pointer to file
Open a File

FILE *f = fopen("filename.txt", "r");
Open a File

FILE *f = fopen("filename.txt", "r");
Open a File

FILE *f = fopen("filename.txt", "r");
Reading from a File

fread(ptr, size, blocks, fp)
Reading from a File

fread(ptr, size, blocks, fp)

Pointer to file to read from
Reading from a File

\texttt{fread(ptr, size, blocks, fp)}

- Pointer to file to read from
- Number of blocks to read
- Pointer to file to read from
Reading from a File

fread(ptr, size, blocks, fp)

Size of a block to read
Number of blocks to read
Pointer to file to read from
Reading from a File

\texttt{fread(ptr, size, blocks, fp)}

- Pointer to memory to read into
- Size of a block to read
- Number of blocks to read
- Pointer to file to read from
Writing to a File

fwrite(ptr, size, blocks, fp)
Writing to a File

fwrite(ptr, size, blocks, fp)

Pointer to file to write to
Writing to a File

fwrite(ptr, size, blocks, fp)

Pointer to file to write to

Number of blocks to write
Writing to a File

\[ \text{fwrite(ptr, size, blocks, fp)} \]

- Pointer to file to write to
- Size of a block to write
- Number of blocks to write
- Pointer to file to write to
Writing to a File

\[ \text{fwrite}(\text{ptr}, \text{size}, \text{blocks}, \text{fp}) \]

- Pointer to data to write
- Size of a block to write
- Number of blocks to write
- Pointer to file to write to
Closing a File

fclose(fp)
Closing a File

fclose(fp)

Pointer to file to close
Other File/IO Functions

- `fgetc`
- `fputc`
- `fprintf`
- `fprintf`
- `...`
Exercise

Write a program `copy.c` that copies a text file.

```
$ ./copy file1 file2
```
Write a program pdf.c that checks if a file is a PDF.
Every PDF begins with the first five bytes: 0x25 0x50 0x44 0x46 0x2d

$ ./pdf file.pdf
YES

$ ./pdf picture.jpg
NO
PART THREE

Dynamic Memory
Dynamic Memory
Dynamic Memory

- Memory that lives on the heap, outside of the stack
- Memory that can be requested at runtime
int *x = malloc(sizeof(int));
```c
int *x = malloc(sizeof(int));
```

Pointer to data on the heap
int *x = malloc(sizeof(int));
int *x = malloc(sizeof(int));
int *x = malloc(sizeof(int));
int *x = malloc(sizeof(int));
int *x = malloc(sizeof(int));
int *x = malloc(sizeof(int));
int *x = malloc(sizeof(int));
int *x = malloc(sizeof(int));

*x = 28;
int *x = malloc(sizeof(int));

*x = 28;
int *x = malloc(sizeof(int));

*x = 28;

free(x);
int *x = malloc(sizeof(int));

*x = 28;

free(x);
int *x = malloc(sizeof(int));
int *x = malloc(3 * sizeof(int));
int *x = malloc(3 * sizeof(int));
int *x = malloc(3 * sizeof(int));
int *x = malloc(3 * sizeof(int));
int *x = malloc(3 * sizeof(int));
int *x = malloc(3 * sizeof(int));

*x = 28;
int *x = malloc(3 * sizeof(int));

*x = 28;
int *x = malloc(3 * sizeof(int));

*x = 28;

*(x + 1) = 42;
int *x = malloc(3 * sizeof(int));

*x = 28;

*(x + 1) = 42;
int *x = malloc(3 * sizeof(int));
*x = 28;
*(x + 1) = 42;
*(x + 2) = 50;
int *x = malloc(3 * sizeof(int));

*x = 28;

*(x + 1) = 42;

*(x + 2) = 50;
int *x = malloc(3 * sizeof(int));
x[0] = 28;
x[1] = 42;
x[2] = 50;
char *s = malloc(3 * sizeof(char));
s[0] = 'H';
s[1] = 'i';
s[2] = '\0';
Memory Errors

- Always free memory after done with it
- Use valgrind to check for memory errors
Problem Set 4
Problem Set 4

- Filter
  - Grayscale
  - Sepia (less comfortable)
  - Reflect
  - Blur
  - Edges (more comfortable)
- Recover
Red 0 255
Green 0 255
Blue 0 255
Red  0  255
Green  0  255
Blue  0  255
| 0xff | 0xd8 |
| 0xff 0xd8 0xff 0xe0 ... | 0xff 0xd8 0xff 0xe1 ... |
This is CS50.