COMPUTER SYSTEMS AND ORGANIZATION

Function Pointers

Daniel G. Graham Ph.D

1. Warmup/Review Memory Leaks
2. Function Pointers

```
1.  // Determine if a number is odd
2.  int isOdd(int *x) {
3.      return (*x) % 2;
4.  }
5.
6.  // Sum up to the first n even numbers
7.  int sumFirstEvens(int *array, int n) {
8.      int *cpy = (int *)malloc(sizeof(n));
9.      int *sum = (int *)malloc(sizeof(int));
10.     int *cpy2 = cpy;
11.     int *sum2 = sum;
12.     for (int i = 0; i < n; i++)
13.         cpy[i] = array[i];
14.     while (!isOdd(cpy)) {
15.         *sum += *cpy;
16.         cpy += 1;
17.     }
18.     free(sum);
19.     return *sum2;
20. }
```

WARM UP

Where should we add free and what should we free.

UNIVERSITY of VIRGINIA | ENGINEERING

WE LOOKED AT THE GENERIC SWAP

WHAT ABOUT A GENERIC SORT

UNIVERSITY of VIRGINIA | ENGINEERING

HOW WOULD THAT FUNCTION PROTOTYPE LOOK LIKE

sort(void *array, size_t num_of_elements, size_t size_of_each element)

Something is missing

Get so we can use memmove to move the elements around, since we number and size.

But how can we compare element

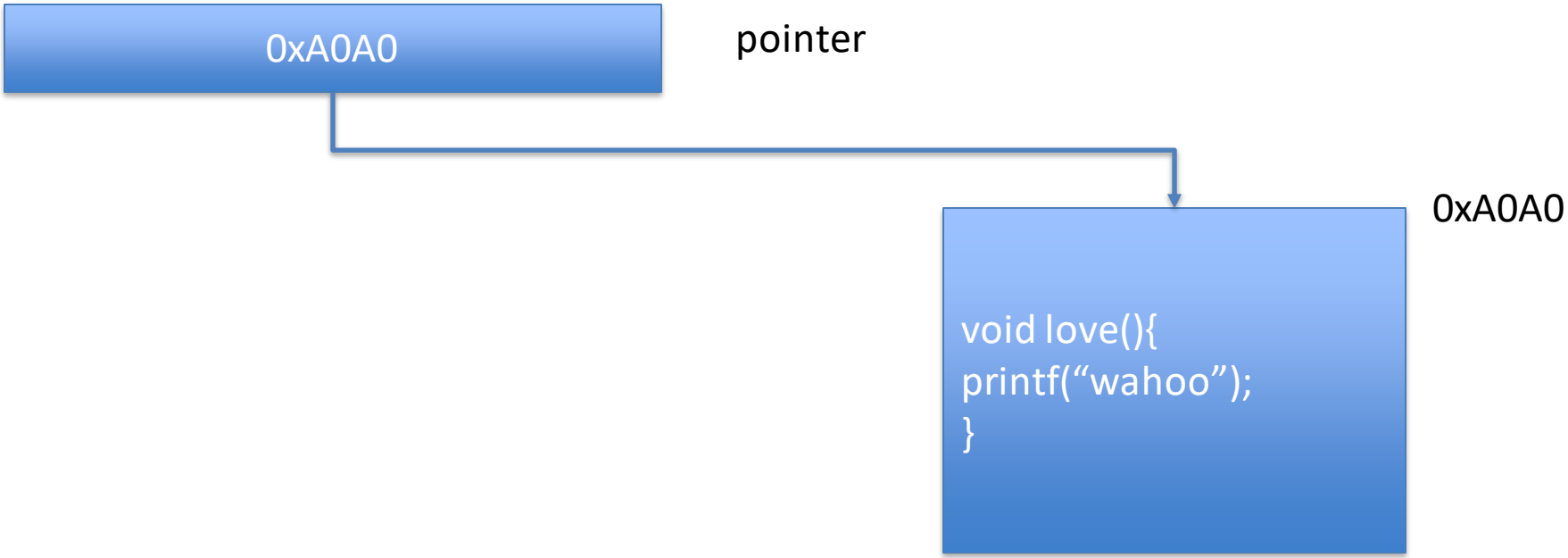
1. Easy for integers and floats
2. But how do we do for structs?

UNIVERSITY of VIRGINIA | ENGINEERING

```
QSORT(3)                    Linux Programmer's Manual                    QSORT(3)

NAME
       qsort, qsort_r - sort an array

SYNOPSIS
       #include <stdlib.h>

       void qsort(void *base, size_t nmemb, size_t size,
                  int (*compar)(const void *, const void *));

DESCRIPTION
       The  qsort() function sorts an array with nmemb elements of size size.  The
       base argument points to the start of the array.
```

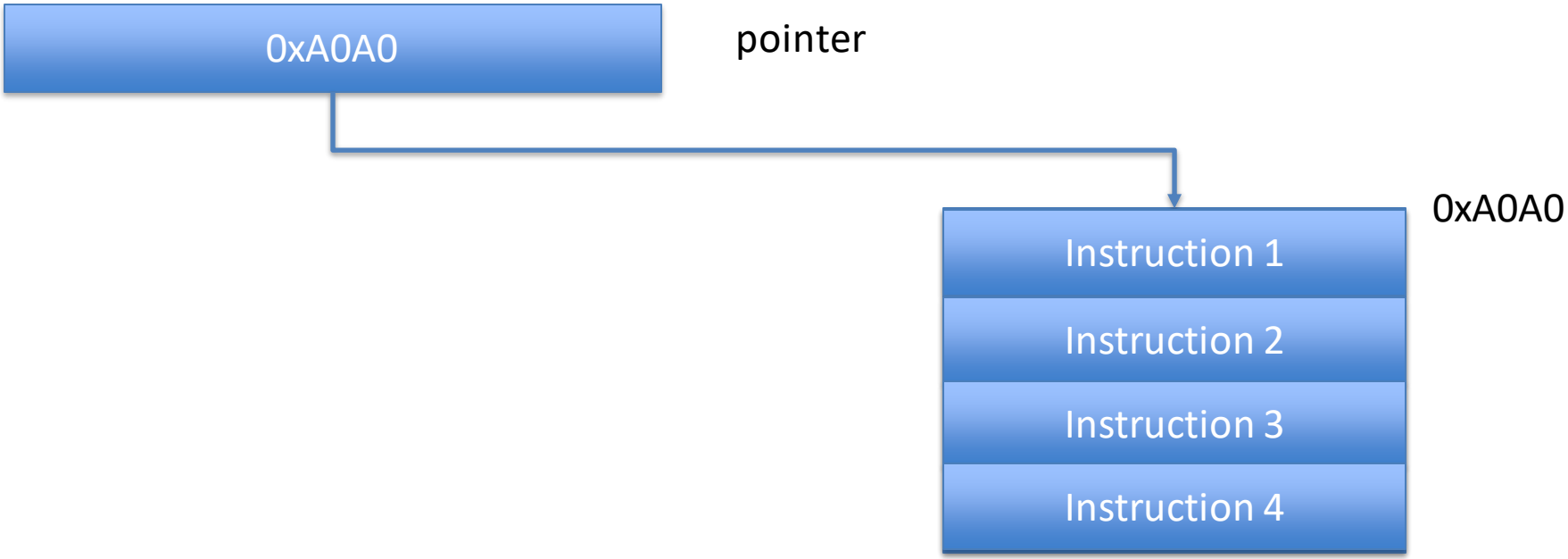

UNIVERSITY of VIRGINIA | ENGINEERING

WHAT IS A FUNCTION POINTER



UNIVERSITY of VIRGINIA | ENGINEERING

WHAT IS A FUNCTION POINTER



UNIVERSITY of VIRGINIA | ENGINEERING

HOW TO DECLARE A POINTER TO A FUNCTION

Need to discuss operator precedence

UNIVERSITY of VIRGINIA | ENGINEERING

OPERATOR PRECEDENCE

How do we declare a variable that is an array of ten integers?

```
int a[10];
```


UNIVERSITY of VIRGINIA | ENGINEERING

OPERATOR PRECEDENCE

How to declare a pointer to an array of ten integers

```
int *ptr[10];
```

Wrong

This will declare an array of 10 elements where each of those elements is a pointer integer, NOT a pointer to an array of 10 integers

UNIVERSITY of VIRGINIA | ENGINEERING

OPERATOR PRECEDENCE

How to declare a pointer to an array of ten integers

```
int *ptr[10];
```

Wrong

[] is higher precedence than *.
So the identifier ptr is associated with []

UNIVERSITY of VIRGINIA | ENGINEERING

OPERATOR PRECEDENCE

How to declare a pointer to an array of ten integers

```
int *ptr[10];
```

Wrong

[] is higher precedence than *.
So the identifier ptr is associated with []

So it is an array of 10 integer pointers

UNIVERSITY of VIRGINIA | ENGINEERING

BUT WE WANT A POINTER TO AN ARRAY OF 10 INTEGERS

```
int (*ptr)[10];
```

So we add brackets around the identifier

UNIVERSITY of VIRGINIA | ENGINEERING

BUT WE WANT A POINTER TO AN ARRAY OF 10 INTEGERS

```
int (*ptr)[10];
```

So we add brackets around the identifier

Now we have a pointer to an array of 10 integers

UNIVERSITY of VIRGINIA | ENGINEERING

BUT WE WANT A POINTER TO AN ARRAY OF 10 INTEGERS

```
int (*ptr)[10];
```

So we add brackets around the identifier

Now we have a pointer to an array of 10 integers

UNIVERSITY of VIRGINIA | ENGINEERING

DECLARING A FUNCTION POINTER

```
float div(int a, int b){
    return a/b;
}
```

```
int main(){
    float (*ptr) ( int, int);
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

DECLARING A FUNCTION POINTER

```
float div(int a, int b){
    return a/b;
}
```

```
int main(){
    float (*ptr) ( int, int);
}
```

pointer

UNIVERSITY of VIRGINIA | ENGINEERING

DECLARING A FUNCTION POINTER

```
float div(int a, int b){
    return a/b;
}
```

```
int main(){
    float (*ptr) ( int, int);
}
```

A function that takes two ints and returns a float

UNIVERSITY of VIRGINIA | ENGINEERING

GENERAL FORM

```
float (*ptr) ( int, int);
```

Return type

UNIVERSITY of VIRGINIA | ENGINEERING

GENERAL FORM

```
float (*ptr) ( int, int);
```

Function name

UNIVERSITY of VIRGINIA | ENGINEERING

GENERAL FORM

```
float (*ptr) ( int, int);
```

parameters

UNIVERSITY of VIRGINIA | ENGINEERING

GENERAL FORM

```
[return type] (*[name])([parameters])
```


UNIVERSITY of VIRGINIA | ENGINEERING

ASSIGNING A FUNCTION POINTER

```
float div(int a, int b){
    return a/b;
}
int main(){
    float (*ptr) ( int, int);
    ptr = &div;
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

USING FUNCTION PTR

```
float div(int a, int b)
{
    return a/b;
}
```

```
int main(){
    float (*ptr) ( int, int);
    ptr = &div;
    float result = (*ptr)(10,20);
    printf("%f", result);
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

USING FUNCTION PTR

```
float div(int a, int b)
{
    return a/b;
}
```

```
int main(){
    float (*ptr) ( int, int);
    ptr = &div;
    float result = (*ptr)(10,20);
    printf("%f", result);
}
```

We don't need &. It is optional for function names since names already represent address.

UNIVERSITY of VIRGINIA | ENGINEERING

USING FUNCTION PTR

```
float div(int a, int b)
{
    return a/b;
}
```

```
int main(){
    float (*ptr) ( int, int);
    ptr = div;
    float result = (*ptr)(10,20);
    printf("%f", result);
}
```

This is valid C code

UNIVERSITY of VIRGINIA | ENGINEERING

USING FUNCTION PTR

```
float div(int a, int b)
{
    return a/b;
}
```

```
int main(){
    float (*ptr) ( int, int);
    ptr = div;
    float result = (*ptr)(10,20);
    printf("%f", result);
}
```

We don't need * or the parentheses. They are also optional.

UNIVERSITY of VIRGINIA | ENGINEERING

USING FUNCTION PTR

```
float div(int a, int b)
{
    return a/b;
}
```

```
int main(){
    float (*ptr) ( int, int);
    ptr = div;
    float result = ptr(10,20);
    printf("%f", result);
}
```

This is also valid c

UNIVERSITY of VIRGINIA | ENGINEERING

ALTERNATIVE

```
float div(int a, int b)
{
    return a/b;
}
```

```
int main(){
    float (*ptr) ( int, int);
    ptr = div;
    float result = *ptr(10,20);
    printf("%f", result);
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

TALK TO YOUR NEIGHBOR

```
#include <stdio.h>

void greet() {
    printf("Hello, World!");
}

int main() {
    void (*funPtr)();
    funPtr = greet;
    (*funPtr)();
    return 0;
}
```

What will happen when I run the following program:

A. Compilation Error
B. Runtime Error
C. Hello, World!
D. No Output

TALK TO YOUR NEIGHBOR

```
#include <stdio.h>

void greet() {
    printf("Hello, World!");
}

int main() {
    void (*funPtr)();
    funPtr = greet;
    (*funPtr)();
    return 0;
}
```

What will happen when I run the following program:

A. Compilation Error
B. Runtime Error
C. Hello, World!
D. No Output

SAME IS TRUE FOR THIS

```
#include <stdio.h>

void greet() {
    printf("Hello, World!");
}

int main() {
    void (*funPtr)();
    funPtr = greet;
    (funPtr)();
    return 0;
}
```

What will happen when I run the following program:

A. Compilation Error
B. Runtime Error
C. Hello, World!
D. No Output

AND THIS

```
#include <stdio.h>

void greet() {
    printf("Hello, World!");
}

int main() {
    void (*funPtr)();
    funPtr = greet;
    funPtr();
    return 0;
}
```

What will happen when I run the following program:

A. Compilation Error
B. Runtime Error
C. Hello, World!
D. No Output

LECTURE

```
#include <stdio.h>

void greet() {
    printf("Hello, World!");
}

int main() {
    void (*funPtr)();
    funPtr = greet;
    *funPtr();
    return 0;
}
```

What will happen when I run the following program:

A. Compilation Error
B. Runtime Error
C. Hello, World!
D. No Output

LECTURE

```
#include <stdio.h>

void greet() {
    printf("Hello, World!");
}

int main() {
    void (*funPtr)();
    funPtr = greet;
    *funPtr();
    return 0;
}
```

What will happen when I run the following program:

A. Compilation Error
B. Runtime Error
C. Hello, World!
D. No Output

Invoke the function then dereference.. Not valid.

TALK TO YOUR NEIGHBOR

Which of the following is the correct way to implement a function that takes a function and calls it

```
int operate(int (*func)(int, int), int x, int y) {
    return func(x, y);
 }
```

```
int operate(int (*func)(int, int) div, int x, int y) {
    return div(x, y);
 }
```

```
int operate(int (*div), int x, int y) {
    return div(x, y);
 }
```


UNIVERSITY of VIRGINIA | ENGINEERING

TALK TO YOUR NEIGHBOR

Which of the following is the correct way to implement a function that takes a function and calls it

```
int operate(int (*func)(int, int), int x, int y) {
    return func(x, y);
 }
```

```
int operate(int (*func)(int, int) div, int x, int y) {
    return div(x, y);
 }
```

```
int operate(int (*div), int x, int y) {
    return div(x, y);
 }
```

The Identifier for function pointers is in the middle.

PASSING A FUNCTION POINTER

```
#include <stdio.h>

int add(int a, int b) {
    return a + b;
}

int operate(int (*func)(int, int), int x, int y) {
    return func(x, y);
}

int main() {
    int (*funcPtr)(int, int) = add;
    int result = operate(funcPtr, 5, 3);
    printf("%d\n", result);
    return 0;
}
```

LET'S LOOK AT AN EXAMPLE FUNCTION THAT WOULD USE A FUNCTION POINTER

UNIVERSITY of VIRGINIA | ENGINEERING

```
QSORT(3)                    Linux Programmer's Manual                    QSORT(3)

NAME
       qsort, qsort_r - sort an array

SYNOPSIS
       #include <stdlib.h>

       void qsort(void *base, size_t nmemb, size_t size,
                  int (*compar)(const void *, const void *));

DESCRIPTION
       The  qsort() function sorts an array with nmemb elements of size size.  The
       base argument points to the start of the array.
```


UNIVERSITY of VIRGINIA | ENGINEERING

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


int main() {
    int arr[] = {10, 5, 15, 3, 12, 7};
    int n = sizeof(arr) / sizeof(arr[0]);

    return 0;
}
```

How do we call qsort

qsort(void *base, size_t nel, size_t width, int (*compar)(const void *, const void *));

UNIVERSITY of VIRGINIA | ENGINEERING

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


int main() {
    int arr[] = {10, 5, 15, 3, 12, 7};
    int n = sizeof(arr) / sizeof(arr[0]);

    // Using qsort to sort the array
    qsort(arr, n, sizeof(int), __________);



    return 0;
}
```

Let's create the function

qsort(void *base, size_t nel, size_t width, int (*compar)(const void *, const void *));

UNIVERSITY of VIRGINIA | ENGINEERING

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

int compareInts(const void *a, const void *b) {

}

int main() {
    int arr[] = {10, 5, 15, 3, 12, 7};
    int n = sizeof(arr) / sizeof(arr[0]);

    // Using qsort to sort the array
    qsort(arr, n, sizeof(int), _______);

    return 0;
}
```

The compare function should subtract b from a and return the result. How would we do this?

qsort(void *base, size_t nel, size_t width, int (*compar)(const void *, const void *));

UNIVERSITY of VIRGINIA | ENGINEERING

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

int compareInts(const void *a, const void *b) {
    return (*(int *)a - *(int *)b);
}

int main() {
    int arr[] = {10, 5, 15, 3, 12, 7};
    int n = sizeof(arr) / sizeof(arr[0]);

    // Using qsort to sort the array
    qsort(arr, n, sizeof(int), compareInts);

    return 0;
}
```

The requirement with the Void* pointer is that we need to Cast the type pointer first

UNIVERSITY of VIRGINIA | ENGINEERING

THERE FUNCTION DECLARATION CAN GET COMPLICATED

```
int (*(*fun_one)(char *,double))[9][20];
```

Are there rules for reading these? Yes ☺

THE RIGHT-LEFT RULE

UNIVERSITY of VIRGINIA | ENGINEERING

The 'right-left' rule simplifies interpreting and creating C declarations.
Symbols:
- '*' means 'pointer to' (left side).
- '[]' means 'array of' (right side).
- '()' means 'function returning' (right side).

Follow these steps:

1. Find the identifier (the variable or function name) and start with ‘<identifier> is'.
2. Check the symbols to the right of the identifier. For example, '()' means 'function returning', and '[]' means 'array of'. Continue right until there are no more symbols or you reach a right parenthesis ')'.
3. Look at the symbols to the left of the identifier. If it's a basic type (like 'int'), state it. Otherwise, use the translations above. Continue left until there are no more symbols or you reach a left parenthesis '(‘.
4. Repeat steps 2 and 3 as necessary

UNIVERSITY of VIRGINIA | ENGINEERING

EXAMPLE 1

```
int *p[];
```

1) Find identifier. int *p[];
                    ^ "p is"

UNIVERSITY of VIRGINIA | ENGINEERING

EXAMPLE 1

2) Move right until out of symbols or right parenthesis hit.

```
int *p[];
      ^^ "p is an array of"
```


UNIVERSITY of VIRGINIA | ENGINEERING

EXAMPLE 1

3) Can't move right anymore (out of symbols), so move left and find:

```
int *p[];
    ^ "p is an array of pointers to"
```


UNIVERSITY of VIRGINIA | ENGINEERING

EXAMPLE 1

4) Keep going left and find:

int *p[];

^^^ "p is an array of pointers to ints".

UNIVERSITY of VIRGINIA | ENGINEERING

EXAMPLE 2

```
int *(*func())();
```


UNIVERSITY of VIRGINIA | ENGINEERING

LAST ONE

```
int (*(*fun_one)(char *,double))[9][20];
```

Removed parameters to make it easier to read

```
int (*(*fun_one)())[9][20];
```


UNIVERSITY of VIRGINIA | ENGINEERING

LAST ONE

```
int (*(*fun_one)(char *,double))[9][20];
```

Removed parameters to make it easier to read

```
int (*(*fun_one)())[9][20];
```

"fun_one is pointer to function expecting (char *,double) and returning pointer to array (size 9) of array (size 20) of int."

NEXT TIME

Returning a function pointer from a function.

Using typedef with function pointers

Right left rule

UNIVERSITY of VIRGINIA | ENGINEERING

REFERENCES

https://www.youtube.com/watch?v=BRsv3ZXoHto

https://cseweb.ucsd.edu/~gbournou/CSE131/rt_lt.rule.html