COMPUTER SYSTEMS AND ORGANIZATION

Function Pointers II

Daniel G. Graham Ph.D

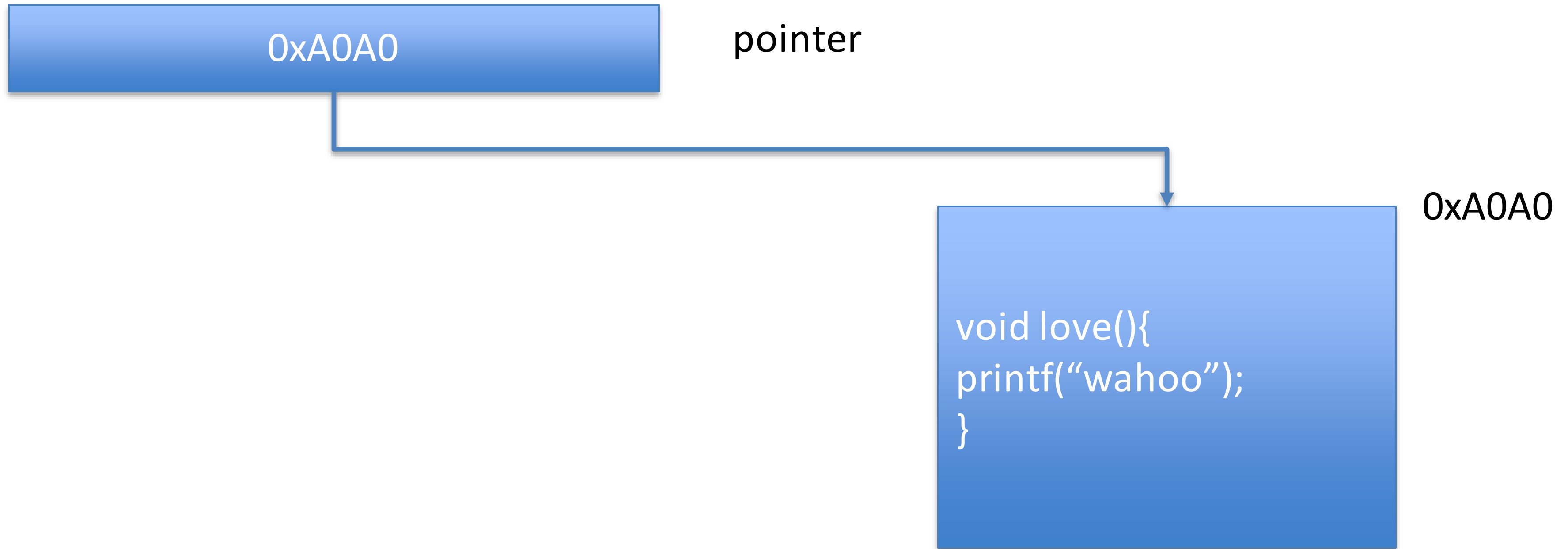
ENGINEERING

1. Function Pointers (Review)
2. Writing functions that return function pointers
3. Type Def Function Pointers
4. Structs And Function Pointers
5. Places where function pointers show up all the time.

FUNCTION POINTERS REVIEW

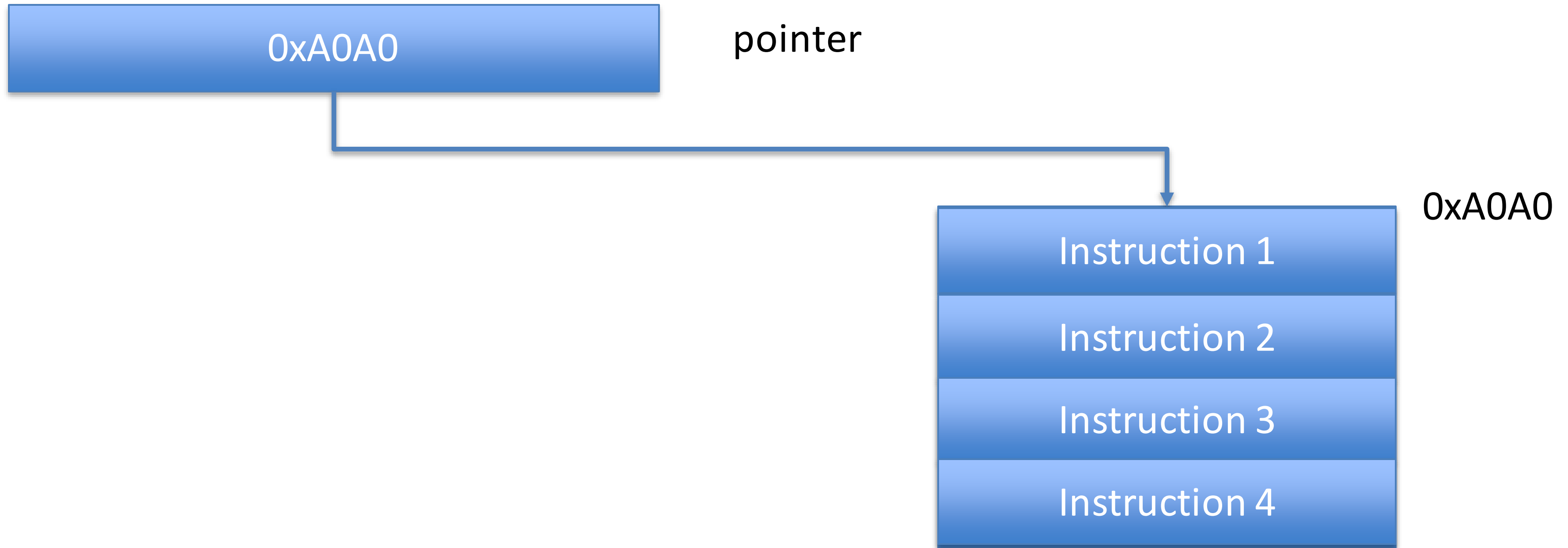
UNIVERSITY of VIRGINIA | ENGINEERING

WHAT IS A FUNCTION POINTER



UNIVERSITY of VIRGINIA | ENGINEERING

WHAT IS A FUNCTION POINTER



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


UNIVERSITY of VIRGINIA | ENGINEERING

USING FUNCTION PTR

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

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


UNIVERSITY of VIRGINIA | ENGINEERING

REVIEW OF RIGHT-LEFT RULE

Write an “English” description of the following declaration.

```
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."

UNIVERSITY of VIRGINIA | ENGINEERING

NORMAL FUNCTIONS

Return type

```
float div(int x, int y ){
    return ((float)x)/y;
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

NORMAL FUNCTIONS

Identifier

```
float div(int x, int y ){
    return ((float)x)/y;
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

BUT FUNCTION POINTERS HAVE THE IDENTIFIER IN THE "MIDDLE"

Identifier

```
float (*return_func)(int, int)
```

Return type

Parameter info

UNIVERSITY of VIRGINIA | ENGINEERING

WRITING FUNCTIONS THAT RETURN FUNCTION POINTERS

```
float div(int x, int y ){
    return ((float)x)/y;
}

float (*return_func())(int, int) {
    return div;
}
```

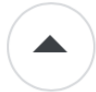
Here is example of function that returns function pointer.

UNIVERSITY of VIRGINIA | ENGINEERING

FUNCTION TYPE

Returning function pointer type

Asked 9 years, 11 months ago Modified 1 year, 5 months ago Viewed 69k times



73

Often I find the need to write functions which return function pointers. Whenever I do, the basic format I use is:

```
typedef int (*function_type)(int,int);

function_type getFunc()
{
   function_type test;
   test /* = ...*/;
   return test;
}
```

They go on to ask the format that we discussed in the previous slides.

LET'S TALK ABOUT TYPEDEF WITH FUNCTION POINTERS

```
typedef float (*function_type)(int, int);

function_type returnDivider(){
    return div;
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

LET'S TALK ABOUT TYPEDEF WITH FUNCTION POINTERS

```
typedef float (*divfunc)(int, int);
```

Identifier

Identifier

```
divfunc returnDivider(){
    return div;
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

WE CAN ALSO TYPE FUNCTION

```
typedef float (*divfunc)(int, int);
```

Identifier

Identifier

```
divfunc returnDivider(){
    return div;
}
```


University of Virginia | ENGINEERING

```
#include <stdio.h>
void myfunction(); //Function prototype

struct MyStruct{
     void (*funcPtr)();
};

void myFunction(){
     printf("Hello from the inside");
}
int main(){
     struct MyStruct example;
     example.funcPtr = myFunction;
     example.funcPtr();
     return 0;
}
```

EXAMPLE 1

UNIVERSITY of VIRGINIA | ENGINEERING

EXAMPLE 2

```
#include <stdio.h>

void onSuccess();
void onError();

struct MyStruct{
	void(*successCallback)();
	void(*errorCallback)();
}

void onSuccess(){
	printf(“Success operation Completed\n”);
}
void onError(){
	printf(“An error occurred. \n”);
}
```

```
void performOperation(struct MyStruct
callbacks, int fail){
	if (!fail){
		callbacks.successCallback();
	}else{
		callbacks.errorCallback();
	}

}

int main(){
	struct MyStruct callbacks;
	callbacks.successCallback = onSuccess;
	callbacks.errorCallback = onError;
	performOperation(callbacks, 1);
	return 0;
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

ENUMS

In C, an enum (short for "enumeration") is a user-defined data type that consists of a set of named integer constants. Enums are used to define a list of named values that represent a finite set of possible values for a variable.

```
enum Name{
    Constant 1,
    Constant 2,
    …
}
```


UNIVERSITY of VIRGINIA | ENGINEERING

ENUMS

```
#include <stdio.h>

enum Day {SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY};

int main() {
     enum Day today;
     today = WEDNESDAY;

    if (today == WEDNESDAY) {
        printf("It's the middle of the week.\n");
    }

    return 0;
}
```

The first element in the array is implicitly assigned the value 0

UNIVERSITY of VIRGINIA | ENGINEERING

ENUMS

```
enum Month {
    January = 1,
    February,
    March,
    April,
    May,
    June,
    July,
    August,
    September,
    October,
    November,
    December
};

int main() {
    enum Month birthMonth;
    birthMonth = April;
    return 0;
}
```

The value of January is explicitly set to 1, and the subsequent months are automatically assigned values incrementing by 1. birth Month is set to April, which would have the value 4

University of Virginia | ENGINEERING

ENUMS

```
enum Direction {
    Up,
    Down,
    Left,
    Right
};

void move(enum Direction dir) {
    switch (dir) {
        case Up:
            // Move up
            break;
        case Down:
            // Move down
            break;
        case Left:
            // Move left
            break;
        case Right:
            // Move right
            break;
    }
}

}
```

```
int main() {
    move(Left);
    return 0;
}
```

Enums make code more readable use them.

University of Virginia | Engineering

VOLATILE AND EXTERN KEY WORDS

UNIVERSITY of VIRGINIA | ENGINEERING

VOLATILE

```
volatile int flag = 0;

void interrupt_handler() {
    // This could be an interrupt handler
    flag = 1;
}

int main() {
    while (!flag) {
        // Waiting for the flag to be set by
the interrupt
    }
    // Proceed once the flag is set
    return 0;
}
```


UNIVERSITY of VIRGINIA ENGINEERING

EXTERN

```
// In some other file, let's say 'file1.c'
int count = 5;

// In the current file
extern int count;

int main() {
    // Now count can be used here
    printf("Count is %d\n", count);
    return 0;
}
```

The extern keyword is used to declare a variable or a function that has external linkage, meaning it is defined in another file or later in the same file. It's a way of declaring the existence of a variable or function without actually creating it.

Prints: “Count is 5”

UNIVERSITY of VIRGINIA | ENGINEERING

STYLE GUIDES AND MORE

https://google.github.io/styleguide/cppguide.html

If you are writing in C++ use smart pointers instead of pointers when ever you can.

UNIVERSITY of VIRGINIA | ENGINEERING

DON’T WORRY, POINTERS HAVE BEEN IMPROVED. WE NOW HAVE SMART POINTERS.

USE SMART POINTERS INSTEAD.

ONLY AVAILABLE IN C++

BIT FIELDS IN C

```
struct
{
    data_type member_name : width_of_bit-field;
};
```

data_type: int, signed int, or unsigned int.

member_name: the name of the bit field.

width_of_bit-field: The number of bits in the bit-field. The width must be less than or equal to the bit width of the specified type

University of Virginia ENGINEERING

BIT FIELDS

```
struct date {
    unsigned int d;
    unsigned int m;
    unsigned int y;
};

int main()
{
    // printing size of structure
    printf("Size of date is %lu bytes\n", aizeof(struct date));
    struct date dt = { 31, 12, 2014 };
    printf("Date is %d/%d/%d", dt.d, dt.m, dt.y);
}
```

```
Size of date is 12 bytes
Date is 31/12/2014
```

BIT FIELDS

```
struct date {
    unsigned int d;
    unsigned int m;
    unsigned int y;
};

int main()
{
    // printing size of structure
    printf("Size of date is %lu bytes\n",
        sizeof(struct date));
    struct date dt = { 31, 12, 2014 };
    printf("Date is %d/%d/%d", dt.d, dt.m, dt.y);
}
```

```
Size of date is 12 bytes
Date is 31/12/2014
```


UNIVERSITY of VIRGINIA | ENGINEERING

BIT FIELDS

```
/ Space optimized representation of the date
struct date {
   // d has value between 0 and 31, so 5 bits are sufficient
   int d : 5;
   // m has value between 0 and 15, so 4 bit are sufficient
   int m : 4;
   int y;
};
int main(){
   printf("Size of date is %lu bytes\n",
        sizeof(struct date));
   struct date dt = { 31, 12, 2014 };
   printf("Date is %d/%d/%d", dt.d, dt.m, dt.y);
   return 0;
}
```

Oh No :(

Size of date is 8 bytes

Date is -1/-4/2014

11111 – Most significant bit is 1 so we have a negative number. (two complement)

UNIVERSITY of VIRGINIA | ENGINEERING

THE FIX

```
/ Space optimized representation of the date
struct date {
   // d has value between 0 and 31, so 5 bits are sufficient
   unsigned int d : 5;
   // m has value between 0 and 15, so 4 bit are sufficient
   unsigned int m : 4;
   int y;
};
int main(){
   printf("Size of date is %lu bytes\n",
          sizeof(struct date));
   struct date dt = { 31, 12, 2014 };
   printf("Date is %d/%d/%d", dt.d, dt.m, dt.y);
   return 0;
}
```

Size of date is 8 bytes

Date is 31/12/2014

No but 8 bytes instead of 5. Why?

University of Virginia | ENGINEERING

LIMITATION WITH BIT FIELDS

1. We can't have pointers to pointers to bit fields

REF

https://www.geeksforgeeks.org/bit-fields-c/?ref=lbp

UNIVERSITY of VIRGINIA | ENGINEERING