

Function Pointers

CS2263 – Systems Software Development

Learning Outcomes

At the conclusion of this lecture students should be able to:

- Describe what a function pointer is
- Explain how a function pointer can be used in C
- Program using the `qsort()` function

More Pointers!

- We can have pointers to variables
- We can have pointers to data structures
 - Homogeneous (like-data types ► arrays)
 - Heterogeneous (dissimilar data types ► structures)
 - *Coming soon!*
- Everything in the machine has an address
- C allows you to access all of it
- Can we have pointers to functions (function pointers)?
 - *Of course we can!*
- WAT?
 - Everything in the machine has an address (remember?)

Functions Have Addresses??

whereIsWaldo.c

```
5  int main(void){
6
7      printf("Address of getStringf(): %p\n", getfString);
8
9      return EXIT_SUCCESS;
10 }
11 /*
12  * [FCSSSDs-iMac-3:Lectures/L15 - Code Along/L15src] wightman% ./whereIsWaldo
13  * Address of getStringf(): 0x102de2d60
14  */
```

- The address of the function in memory
- What happens when we call a function?
 - Push space for the return value (if required) onto the stack
 - Push the address we're going to process next
 - Push the arguments
 - Change the address to process next to the function's starting address

Example Function

functionPointer.c

```
22  int compareInt(const void * arg1, const void * arg2){
23      // convert void * to an int *
24      const int * ptr1 = (const int *) arg1;
25      const int * ptr2 = (const int *) arg2;
26      // get the value from the address
27      const int val1 = * ptr1;
28      const int val2 = * ptr2;
29      // compare the value
30      if (val1 < val2) return -1;
31      if (val1 == val2) return 0;
32      return 1;
33  }
```

Example Use

functionPointer.c

```
4  int compareInt(const void * arg1, const void * arg2);
5
6  // Note - this program *should* do error checking!!
7  int main(void){
8      int a;
9      int b;
10     int* pa = &a;
11     int* pb = &b;
12
13     printf("Two integers please: ");
14     scanf(" %d %d", pa, pb);
15
16     if(compareInt(pa, pb)) printf("Integers are in ascending order\n");
17     else printf("Integers are equal or in descending order\n");
18
19     return EXIT_SUCCESS;
20 }
```

```
34  /*
35   * [FCSSSDs-iMac-3:Lectures/L15 - Code Along/L15src] wightman% ./functionPointer
36   * Two integers please: 3 4
37   * Integers are in ascending order
38   * [FCSSSDs-iMac-3:Lectures/L15 - Code Along/L15src] wightman% ./functionPointer
39   * Two integers please: 4 3
40   * Integers are in ascending order
41   * /
42
```

qsort(): The Example

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

- Sorts array of `nel` elements, each of `width` bytes, using function `compar(const void *, const void *)` to compare elements.
- Found in `<stdlib.h>`
- One of the arguments is an integer function pointer:
`int (*compar)(const void *, const void *)`
 - Takes two pointers
 - Returns an integer indicating order:
 - -1 if `arg1 < arg2`
 - 0 if `arg1 == arg2`
 - 1 if `arg1 > arg2`
- Allows programmer-defined collation sequence (sorting order)
 - *user-defined data types are coming!*

Example Use

sortInts.c

```
7 // Note - this program *should* do error checking!!
8 int main(void){
9     int a[MAX_SIZE];
10
11     printf("%d integers please: ", MAX_SIZE);
12     for(int i=0; i< MAX_SIZE; i++){
13         scanf("%d", &a[i]);
14     }
15
16     qsort(a, MAX_SIZE, sizeof(int), compareInt);
17     printf("Sorted values: ");
18     for(int i=0; i< MAX_SIZE; i++){
19         printf("%d ", a[i]);
20     }
21     printf("\n");
22
23     return EXIT_SUCCESS;
24 }
25
26
27
28 /*
29  * [FCSSSDs-iMac-3:Lectures/L15 - Code Along/L15src] wightman% ./sortInts
30  * 10 integers please: 5 34 -1 0 444 34 -34 5 34 -66
31  * Sorted values: -66 -34 -1 0 5 5 34 34 34 444
32  */
33
34
35
36
37
38
39
40
41
42
43
```