Lecture #15 – More Function Pointers, const, structs
Recap

• Generics So Far
• Motivating Example: Bubble Sort
• Function Pointers
Recap: Generics Overview

• We use `void *` pointers and memory operations like `memcpy` and `memmove` to make data operations generic.

• We use `function pointers` to make logic/functionality operations generic.
Plan for Today

• Function Pointers (cont’d.)
• const

Disclaimer: Slides for this lecture were borrowed from
—Nick Troccoli's Stanford CS107 class
Lecture Plan

• Function Pointers (cont’d.)
  • const
Function Pointers

• Function pointers can be used in a variety of ways. For instance, you could have:
  • A function to compare two elements of a given type
  • A function to print out an element of a given type
  • A function to free memory associated with a given type
  • And more...
Function Pointers

• Function pointers can be used in a variety of ways. For instance, you could have:
  • A function to compare two elements of a given type
  • A function to print out an element of a given type
  • A function to free memory associated with a given type
  • And more...
Demo: Generic Printing

print_array.c
Common Utility Callback Functions

• Comparison function – compares two elements of a given type.

  \[
  \text{int} \ (\ast \text{cmp}_\text{fn})(\text{void} \ *\text{addr}1, \ \text{void} \ *\text{addr}2)
  \]

• Printing function – prints out an element of a given type

  \[
  \text{void} \ (\ast \text{print}_\text{fn})(\text{void} \ *\text{addr})
  \]

• There are many more! You can specify any functions you would like passed in when writing your own generic functions.
Demo: Count Matches

count_matches.c
Count Matches

• Let’s write a generic function `count_matches` that can count the number of a certain type of element in a generic array.

• It should take in as parameters information about the generic array, and a function parameter that can take in a pointer to a single array element and tell us if it’s a match.

```c
int count_matches(void *base, int nelems,
                  int elem_size_bytes, bool (*match_fn)(void *));
```
Count Matches

```c
int count_matches(void *base, int nelems, int elem_size_bytes,
        bool (*match_fn)(void *)) {

    int match_count = 0;

    for (int i = 0; i < nelems; i++) {
        void *curr_p = (char *)base + i * elem_size_bytes;
        if (match_fn(curr_p)) {
            match_count++;
        }
    }

    return match_count;
}
```
Function Pointers As Variables

In addition to parameters, you can make normal variables that are functions.

```c
int do_something(char *str) {
    ...
}

int main(int argc, char *argv[]) {
    ...
    int (*func_var)(char *) = do_something;
    ...
    func_var("testing");
    return 0;
}
```
Generic C Standard Library Functions

- **qsort** – I can sort an array of any type! To do that, I need you to provide me a function that can compare two elements of the kind you are asking me to sort.

- **bsearch** – I can use binary search to search for a key in an array of any type! To do that, I need you to provide me a function that can compare two elements of the kind you are asking me to search.

- **lfind** – I can use linear search to search for a key in an array of any type! To do that, I need you to provide me a function that can compare two elements of the kind you are asking me to search.

- **lsearch** – I can use linear search to search for a key in an array of any type! I will also add the key for you if I can’t find it. In order to do that, I need you to provide me a function that can compare two elements of the kind you are asking me to search.
Generic C Standard Library Functions

• `scandir` – I can create a directory listing with any order and contents! To do that, I need you to provide me a function that tells me whether you want me to include a given directory entry in the listing. I also need you to provide me a function that tells me the correct ordering of two given directory entries.
Summary: Function Pointers

• We can pass functions as parameters to pass logic around in our programs.

• Comparison functions are one common class of functions passed as parameters to generically compare the elements at two addresses.

• Functions handling generic data must use *pointers to the data they care about*, since any parameters must have *one type* and *one size*. 
Lecture Plan

• Function Pointers (cont’d.)

• const
• Use `const` to declare global constants in your program. This indicates the variable cannot change after being created.

    const double PI = 3.1415;
    const int DAYS_IN_WEEK = 7;

    int main(int argc, char *argv[]) {
      ...
      if (x == DAYS_IN_WEEK) {
        ...
      }
      ...
    }
const

• Use `const` with pointers to indicate that the data that is pointed to cannot change.

```c
char str[6];
strcpy(str, "Hello");
const char *s = str;

// Cannot use s to change characters it points to
s[0] = 'h';
```
const

Sometimes we use `const` with pointer parameters to indicate that the function will not / should not change what it points to. The actual pointer can be changed, however.

```c
// This function promises to not change str’s characters
int countUppercase(const char *str) {
    int count = 0;
    for (int i = 0; i < strlen(str); i++) {
        if (isupper(str[i])) {
            count++;
        }
    }
    return count;
}
```
const

By definition, C gets upset when you set a non-\texttt{const} pointer equal to a \texttt{const} pointer. You need to be consistent with \texttt{const} to reflect what you cannot modify.

\begin{verbatim}
// This function promises to not change str’s characters
int countUppercase(const char *str) {
    // compiler warning and error
    char *strToModify = str;
    strToModify[0] = ...;
}
\end{verbatim}
const

By definition, C gets upset when you set a non-const pointer equal to a const pointer. You need to be consistent with const to reflect what you cannot modify. Think of const as part of the variable type.

// This function promises to not change str’s characters
int countUppercase(const char *str) {
    const char *strToModify = str;
    strToModify[0] = ...
}
**const**

**const** can be confusing to interpret in some variable types.

```cpp
// cannot modify this char
const char c = 'h';

// cannot modify chars pointed to by str
const char *str = ...;

// cannot modify chars pointed to by *strPtr
const char **strPtr = ...;
```
Practice
Which lines (if any) above will cause an error due to violating `const`?
Remember that `const char *` means that the characters at the location it stores cannot be changed.
Which lines (if any) above will cause an error due to violating `const`?
Remember that `const char *` means that the characters at the location it stores cannot be changed.
Which lines (if any) above will cause an error due to violating `const`?
Remember that `const char *` means that the characters at the location it stores cannot be changed.
Which lines (if any) above will cause an error due to violating `const`? Remember that `const char *` means that the characters at the location it stores cannot be changed.
Which lines (if any) above will cause an error due to violating `const`? Remember that `const char *` means that the characters at the location it stores cannot be changed.
const

1. char buf[6];
2. strcpy(buf, "Hello");
3. const char *str = buf;
4. str[0] = 'M';
5. str = "Mello";
6. buf[0] = 'M';

Line 5 is ok – str's type means that while you cannot change the characters at which it points, you can change str itself to point somewhere else. str is not const – its characters are.

Which lines (if any) above will cause an error due to violating const?
Remember that const char * means that the characters at the location it stores cannot be changed.
Which lines (if any) above will cause an error due to violating `const`?

Remember that `const char *` means that the characters at the location it stores cannot be changed.

Line 6 is ok – `buf` is a modifiable char array, and we can use it to change its characters. Declaring `str` as `const` doesn't mean that place in memory is not modifiable at all – it just means that you cannot modify it using `str`. 
Recap

• Function Pointers (cont’d.)
• const

Next Time: Structs