Function Pointers

Special Topic

Function Pointers

- A **pointer** is a variable that holds the address of another variable
- Can also have **function pointer**, which hold a reference to a particular function
  - Like a data type for holding a reference to a function
- A function pointer can be used to call the function it points to, much like how a pointer can be used to access the value it points to

- Pointers have data types indicating what type of data they're pointing to
- Function pointers similarly specify the return type and parameter types of the function they point to

- Syntax for making a function pointer is kinda like making a function prototype:
  - ***returnType* (**functionPointerName*)(*parameters*)**

- Examples:
  - **void (*signalHandler)(int)**
    - **signalHandler** is a pointer to a void function that has a single integer parameter
  - **void (*displayFunction)(void)**
    - displayFunction is a pointer to a void function that has no parameters

- **void (*compareFunction)(const void *, const void *)**
  - **compareFunction** is a pointer to a void function that has two constant void pointers as inputs
- **double (*averageFunction)(double *, int)**
  - **averageFunction** is a pointer to a function that returns a double and has an input that is a pointer to a double (for example, an array) and an int

- A function can be put in the function pointer by setting the function pointer equal to the name of the function
- Suppose there's a function pointer has been created as well as a function f(), then:
  - **functionPointer = f;**
- Also can put & in front of the function name, but not required
  - **functionPointer = &f;**

- Once a function reference has been put in the function pointer, we can treat the function pointer like it was the name of the function
- If the function has one integer parameter, then we could call it with the function pointer:
  - **functionPointer(5);**

- **function_pointer1.c**
  - Creates a few functions
  - Makes a function pointer
  - Puts the function in it
  - Calls the function by using the function pointer

- Function pointers allow us to be able to pass a function as an input into another function
  - Can be used to achieve polymorphism / generic
- Function pointers can also be used for callback functions
  - Very useful for implementing certain design patterns
    - Event-driven programming

qsort() and bsearch()

- Two specific examples of using function pointers are **qsort()** and **bsearch()**
- Designed to work on any data type
- To use these functions, we need to create a function that compares two values

Compare Function

- A compare function takes in pointers to two items
- The compare function returns either 0, negative, or positive indicating how the two items compare to each other

- Remember strcmp()
  - 0 means item a equal item b
  - -1 means item a is less than item b
  - 1 means item a is greater than item b

- It's up to us to define how these two items relate to one another
  - We just need to return 0, -1, or 1 from our compare function
    - Actually, "-1" can be any negative integer and "1" can be any positive integer

- Allows us to define how items of some custom data types relate to each other
  - What does it mean for two employee_t variables to be equal, less than, or greater than?

- A compare function must have a particular signature
  - Return type is int
  - Has two const void * parameters
    - void * is so it can work with any data type
    - const is to make the pointer a constant

```
int compare_function(const void *a, const void *b)
{
    /* do the comparison */
}
```

- After we give our compare function to qsort() or bsearch(), they will automatically call the compare function

qsort()

- **qsort()** is a function for sorting an array
  - Need to include **stdlib.h** to use
- Designed to work on arrays of any data type
- Uses compare function on elements of the array as it sorts

- qsort() has several inputs
  - Array variable (or pointer to beginning of array)
  - Array's length
  - Size of each element in the array (use sizeof())
  - Compare function

```
void qsort(void *array,
    int arrayLength,
    int elementSize,
    int (*compareFunction)(const void *, const void *))
```

```c
int compare_function(const void *a, const void *b)
{
    return *(int *)a - *(int *)b;
}

/* ... */

int data[100];

/* ... */

qsort(data, 100, sizeof(int), compare_function);
```

bsearch()

- **bsearch()** is a function for searching an array
  - Need to include **stdlib.h** to use
- Array must be sorted in ascending order for **bsearch()** to work properly
  - Requirement of binary search algorithm

- bsearch() has several inputs
  - Item searching for (called the key)
  - Array variable (or pointer to beginning of array)
  - Array's length
  - Size of each element in the array
  - Compare function

- If bsearch() finds the item we're looking for, its output is a pointer to that element in the array
  - We can determine the index of this element by subtracting array from the result
  - result - array = index
- If bsearch() does not find the item, the output is a NULL pointer

```
void * bsearch(
    void *key,
    void *array,
    int arrayLength,
    int elementSize,
    int (*compareFunction)(const void *, const void *)
)
```

```c
int compare_function(const void *a, const void *b)
{
   return *(int *)a - *(int *)b;
}

/* ... */

int data[100];
int key = 42;
int * result;

/* ... */

qsort(data, 100, sizeof(int), compare_function);

result = bsearch(&key, data, 100, sizeof(int), compare_function);
```

- **qsort_bsearch_example.c**
  - Creates an array of random integers
  - Uses qsort() on array
  - Searches array using bsearch()
    - Checks if result is NULL and prints result if not