An introduction to function pointers: Part 1

Jacob Beningo - January 15, 2013

A critical tool when developing an embedded system in C is an understanding of how pointers work. Misunderstanding pointer fundamentals can result in long hours spent debugging a system or a hidden bug that reveals itself in front of the customer. One of the pointer tools that is most often misunderstood and ignored is function pointers.

Function pointers are identical to any other pointer type except that they point to a function instead of a variable type. The pointer can then be used to call the function that is being pointed to.

This allows a whole range of techniques to be used when developing an embedded system. For example, function pointers can be used in task schedulers to dynamically call which function is executed from a task table. They can be used in complex state machines or switch statement to decrease overhead of a decision and to break up states into manageable and maintainable chunks.

Function pointers frequently get a bad rap due to fear of complexity and lack of understanding. This has even come to the point of some companies even outlawing their use! Despite the drastic measures some developers go to in order to avoid function pointers, they are really a tool that every developer should have in their tool bag. This and the next several posts will explore how to define function pointers, different application purposes they serve and how to safely use them.

The declaration of a function pointer is very similar to the declaration of any other function. It requires a return type, declaration name and parameter list. The major difference in declaring a function pointer is that the declaration name is preceded by a * and then must be enclosed by parentheses. Failure to enclose the declaration name in parentheses will result in a compiler error. Listing 1 shows an example function pointer declaration for a simple function that takes no parameters and returns nothing.

```
void (*FuncPtr)(void);
```

Listing 1. Function pointer declaration for void function

Listing 1 demonstrates the simplest definition for a function pointer but any combination of function pointer types can be declared. For example, **Listing 2** shows the definition of additional function pointers with increasing complexity. Keep in mind

that the function pointer is declared with the same parameters as the function that it is pointing to! That means there are nearly an infinite number of possible function pointer definitions.

```
void (*FuncPtr)(int);
int (*FuncPtr)(int);
int (*FuncPtr)(int, char *);
int * (*FuncPtr)(int, int, char, char *);
int * (*FuncPtr)(int, int, char, void (*FuncPtr2)(void));
```

Listing 2. Function pointer declaration examples

The notation for declaring a function pointer can look pretty ugly and complicated; however, there are instances when the same pointer declaration will be reused over and over again. When a function pointer needs to be declared of the same notation multiple times, it is useful to declare the function pointer using a typedef. This allows a simpler declaration to be used to define the function pointer. **Listing 3** shows an example of how to create the typedef.

```
typedef int * (*FuncPtr)(int *);
```

Listing 3. Type declaration of a function pointer

With the typedef it becomes fairly straight forward to declare new function pointers and set them. In fact, it begins to look exactly like any other pointer declaration. **Listing 4** shows the declaration of NewPtr and then sets it to point to Function1.

```
FuncPtr NewPtr;
NewPtr = &Function1;
```

Listing 4. Declaration of Function Pointer Method 1

There are two different ways in which the pointer can be set. The first was shown in Listing 4, the second is shown in **Listing 5**. The two methods can be used interchangeably and only differ in the use of the & (get address of) operator.

```
FuncPtr NewPtr;
NewPtr = Function1;
```

Listing 5. Declaration of Function Pointer Method 2

There are two different calling methods that can be used to execute the pointed to function. The first looks exactly like calling a normal function as can be seen in **Listing 6.** The second looks similar to the function pointer declaration. It requires the use of *, to dereference the address of the pointer and then once again the pointer is enclosed with parentheses. The second method is shown in **Listing 7**.

```
Result = NewPtr(intPtr);
```

Listing 6. Calling method 1

```
Result = (*NewPtr)(intPtr);
```

Listing 7. Calling method 2

Either method works to call the function but it is helpful to use the second method. It does look more complicated but it is then obvious that NewPtr is a function pointer and not just a regular function. This ensures that if the pointer isn't named in an obvious way that during maintenance there isn't a misunderstanding.

When using function pointers it is absolutely important that before the function pointer is called, it be checked to make sure that the pointer has been set. The reason for this is that if the pointer has not been set the application will begin executing code at the location it is pointing to! If the pointer is pointing out in the weeds then the system will most likely crash!

The minimum that can be checked is that the pointer is not equal to zero prior to executing the function! This is shown in **Listing 8.** Depending on the application, there may be additional checks that can be performed to verify that the value stored in the pointer is correct and should be executed.

```
if(NewPtr != 0U)
{
   Result = (*NewPtr)(intPtr);
}
```

Listing 8. Check that the pointer is valid before executing

Function pointers are an extremely useful tool. This article examined how to define and use function pointers from a very basic stand point. In the next few posts, common uses for function pointers will be surveyed starting with task scheduling.

To read **Part 2** in this series, go to "**Task scheduling**."
To read **Part 3** in this series, go to "**State machines.**"

This article was previously published on Embedded.com's sister publication, ***EDN Magazine****.*