



Lecture 7

Neat C Techniques

AAYUSH ACHARYA
KANCHAN POUDEL



Function Pointers

Point to the address of the function

Way to assign a function to a variable

Useful to pass functions as argument to another function, return functions from subroutine



```
#include<stdio.h>
```

```
// function to see address of main function.
```

```
int main()
```

```
{
```

```
    printf("Address of main function is %p", main);
```

```
    getchar();
```

```
    return 0;
```

```
}
```



Declaration of function Pointer

```
returnType (* pointerName) (param1Type, param2Type...)
```

Can assign any function with the same return type and parameters to the pointer


Example:

```
int (*ptr)(int, int);
```

```
int func (int, int);
```

```
ptr = func
```

ptr contains address of func.



Declaration of a function pointer should have the pointer name in the parentheses as function parameters have precedence over pointers in declarations.

```
int * ptr(int, int) //declaration of function, named ptr returning an  
int*
```



Typedef for function pointers

Using typedef can simplify the usage of function pointers

Helps to name a type of function pointer, associated with a function signature of function with a particular return type and function parameters

Example Code: [typedef](#)



Callbacks

The callback function is used when a reference of a “function 1” is passed as an argument to “function 2” using a function pointer.

A callback function is simply a function pointer that is passed to another function as a parameter. In most instances, a callback will contain three pieces:


- The callback function
- A callback registration
- Callback execution

Example Code: [simpleCallback](#), [callback2](#)



Polymorphism using void pointers

Polymorphism allows a program to process objects differently depending on data type it receives as properties or parameters.



The `qsort()` standard library function:

```
void qsort (  
    void *base,                /* Array to be sorted */  
    size_t num,                /* Number of elements in array */  
    size_t size,              /* Size in bytes of each element */  
    int (*compare)(const void *, const void *); /* Comparison function for two elements */
```

In order to use this as a generic sorting function with polymorphism, void pointers are used.

Since the input array parameters in compare function are void pointers, we can cast them as we need to sort the array of data type of our wish.

If we wish to sort an array of floats, the code in the following slide shows how compare function is implemented for comparing two floats.

Then if we call `qsort()` with input array of floats and the given compare function, it will work to sort array of floats and similar for int or char.



```
int compare_floats(const void *a, const void *b)
```

```
{float fa = *((float *)a);
```

```
float fb = *((float *)b);
```

```
if (fa < fb)
```

```
    return -1;
```

```
if (fa > fb)
```

```
    return 1;
```

```
return 0;}
```

```
qsort(array, len, sizeof(array[0]), compare_floats);
```

```
}
```

Example Code: [virtualPolymorphism](#)



References

- [Function Pointers and Callbacks in C - An Odyssey](#)
- [C Language Tutorial ⇒ Polymorphic behaviour with void pointers](#)