Pointers

A pointer is a variable that holds the memory address of another variable, constant or literal. They allow us to perform a call by reference in a way that variables outside the scope of the function can be modified within such function. They are also useful for managing strings as an array of characters, where the pointer can "point" to the beginning of such array or any specific character within the array.

Declaring a Pointer Type

Declaration:

*type * identifier;* Where *type* can be any atomic (int, char, etc) or a custom type previously defined using a typedef, and *identifier* is the name of the pointer variable.

We say that the actual type of the pointer is not *type,* but *type **. For instance:

Int pMyInt* is a pointer to a variable of type *int,* therefore the type of such pointer is *int** and holds the address of an *int* variable.

Assigning a pointer

Once we have declared a pointer, we can then assign an address of a variable or constant / literal to it. To do that, we use the address *(&)* operator. For instance:

```
Int MyInt;

Int* pMyInt = &MyInt;
```

Obtaining the Content of the Variable Pointed

Once we have declared and assigned a pointer, we can now reference the variable using the pointer name with asterisk in front of it, in which case we can read such variable or assign a new value to it. For instance:

```
Int MyInt;

Int* pMyInt = &MyInt;

*pMyInt = 5; // Assigns 5 to the variable being pointed by pMyInt, MyInt

Int sum = *pMyInt + 2 //Assigns 5+2 into sum
```

Using Pointers to Structures

We can access the structure through the pointer, just like any variable, then we can reference the structure member using the accessor: "." Operator. For instance:

```
typedef struct MyStruct_
{
        int Member1;
        float Member2;
} MyStruct;
MyStruct record;
MyStruct * pMyStruct = &record;
*pMyStruct.Member1 = 5;
*pMyStruct.Member2 = 2.5;
```

Since accessing structure elements via pointers is common and useful, an alternative and easier to understand syntax is available:

```
pMyStruct->Member1 = 5;
pMyStruct->Member2 = 2.5;
```

Operations with pointers

Since a pointer variable holds an address, it therefore holds a number, which could be incremented or decremented, meaning that incrementing such pointer would make it point to the next address, or decrementing it, would make it point to the previous address; the addresses would be incrementing or decrementing a number of bytes equal to the size of the variable that is being pointed. For instance, a pointer to a *char* variable would increment /decrement by 1 byte, while a pointer to an *int* (16 bytes) would make increments / decrements of 2 bytes.

This type of operation can be extremely dangerous as we could make the pointer point to a wrong address by mistake, so caution must be taken with using pointer arithmetic.