C Programming Basics

Adam Montgomery

14 February 2019

Data Types

Numeric

int Integers, usually 4 bytes
float Floating point numbers, decimals
double Double precision decimal numbers
long Integer number with more space
short Integer with less space

Characters

char Character or letter 1 byte
strings Array of characters, null terminated

Typedefs

We can use a typedef statement to define a new datatype. This is usually used for things like structs where we don't want to continue to type 'struct' before the variable name.

```
typedef struct Person{

    char *name;
    unsigned char age;
    int weight;

}Person;

Person p;
p.name = "Adam";
p.age = 30;
p.weight = 240;
```

Enums

Enumerations can be useful for creating flags, and options to pass to functions. Enums assign values starting at zero, unless you assign your own value to the elements of the enum.

```
enum Week{Mon, Tues, Wed, Thurs, Fri, Sat, Sun};

enum Week day = Tues; // day=1
```

```
enum Year{Jan=1, Feb, Mar, Apr};
```

Data Structures

Data structures are a means of storing variables efficiently, and logically.

Arrays

Arrays are the simplest type of data structure in C. They are collections of elements of the same data type. All of the primitive data types such as 'int', as well as any other user defined data type may be placed in an array. Arrays zero-indexed, meaning that the first element is the 0th element.

Structs

Structs are similar to objects in other programming languages. They may hold member variables, and be passed into functions either by value or by reference.

Pointers

Pointers can be difficult to manage or even understand. Just keep in mind that they are just memory addresses.

Function Pointers

Function pointers can be used to pass functions to other functions in order to modify operation of the calling function. A classic example is to pass a function pointer to a sorting function.

```
#include <stdlib.h>

int int_sorter( const void *first_arg, const void *second_arg ){
    int first = *(int*)first_arg;
    int second = *(int*)second_arg;
    if ( first < second ){
        return -1;
    }
    else if ( first == second ){
        return 0;
    }
    else{
        return 1;
    }
}

int main(){
    int array[10];
    int i;
        /* fill array */
    for ( i = 0; i < 10; ++i ){
        array[ i ] = 10 - i;
```

```
    }
    qsort( array, 10 , sizeof( int ), int_sorter );
    for ( i = 0; i < 10; ++i ){
        printf ( "%d\n" ,array[ i ] );
    }
}
```

This code is taken from cprogramming.com.

Access Structure Members

As with any variable, many times we will want to pass by reference instead of passing by value. Passing pointers to structs is no different. There is a difference in how a struct's member variables are accessed though. The arrow `->` operator serves the function of dereferencing the struct and accessing the member at the same time.

```
struct Node{
    int value;
    struct Node next_node;
}

struct Node *n = malloc(sizeof(struct Node));

// dereference and access member
(*n).value = 10;

// use arrow operator instead
n->value = 10;
```

Storage Classifiers

Const

Denotes that a variable is constant, immutable. Compiler will not allow the programmer to change the value once it is set. We can use this for naming memory locations as well because the address will not change so we make a const pointer to an address.

Static

Static has 2 different contexts in C. Placing static before a variable name ensures that that varable doesn't go out of scope. Static variables inside functions hold their value after the function returns and goes out of scope.

The other use of static is to limit a global variable or function's scope to the file it is declared in. This is useful in making functions in a library 'private'.

Volatile

The volatile keyword lets the compiler know that the value may change at any time. Used mainly in programs where the control flow is not linear, such as anytime there are interrupts involved.

Extern

The extern keyword prevents memory from being allocated for a variable since it is implied that the 'definition' of the variable or function is done elsewhere. Extern also extends the scope of a variable or function to the entire program not only the compilation unit of which it is a part.