C Programming: Recap

Pointer Declarations

```
char *pc;
unsigned int *pi;
long *pl;
```

Dereferencing

```
char c = *pc;
unsigned int i = *pi;
long l = *pl;
```

Pointer Arithmetic

You may perform arithmetic on pointers:

```
pc + 5
pi + 5
pl + 5
```

But you have to remember that the compiler takes the size of the underlying type into account. So if pc starts with value 2000, the expression pc + 5 returns 2005. However, if pi starts with value 2000, the expression pi + 5 returns 2020 (assuming sizeof(int) == 4), while if pl starts with value 2000, the expression pl + 5 returns 2040 (assuming sizeof(long) == 8).

Increment & Decrement Operators

- Unary operators (i.e., they take exactly 1 operand)
- ++ increments the operand by one
- −− decrements the operand by one
- Pre-increment: operator appears before the operand (e.g., ++i)
- Post-increment: operator appears after the operand (e.g., i++)
- Pre-decrement: operator appears before the operand (e.g., −−i)
- Post-decrement: operator appears after the operand (e.g., i−−)
- Pre-increment & pre-decrement: value of the expression based on the *incremented/decremented* operand value

- Post-increment & post-decrement: value of the expression based on the *original* operand value
- Increment and decrement operators on pointers: pointer arithmetic applies
- Contrast *++p (modifies p & returns the value at the incremented address) vs. *p++ (modifies p *but* returns the value at the original address)

Bit-Level Operations in C

One useful feature of C is that it supports bitwise Boolean operations, also know as bit vector operations.

There are six such operators:

- ~: applies the Boolean NOT operation on each bit of the operand (e.g., ~0x0F0F0F0Fu == 0xF0F0F0F0u)
- | : applies the Boolean OR operation on pairs of bits of the operands (e.g., 0x12345678u | 0x0FFF0000u == 0x1FFF5678u)
- & : applies the Boolean AND operation on pairs of bits of the operands (e.g., 0x12345678u | 0x0FFF0000u == 0x02340000u); warning: this is a case of an overloaded operator (i.e., an operator that means different things in different contexts), in some contexts, it can also mean "take the address of"
- ^ : applies the Boolean EXCLUSIVE-OR on pairs of bits of the operands (e.g., 0x12345678u ^ 0x0FFF0000u == 0x1DCB5678u)
- << : shifts the left operand by the number of bit positions indicated on the right (e.g., 0x12345678u << 4 == 0x23456780u); it is often the same as multiplying the left operand by 2^n, where *n* is the value of the right operand
- >> : shifts the left operand by the number of bit positions indicated on the right (e.g., 0x12345678u >> 4 == 0x01234567u); it is often the same as dividing (rounded down) the left operand by 2^n, where *n* is the value of the right operand; beware when applying this operator on signed types

You might ask what is the use of these operators. Some times programmers store information as bit vectors where some groups of bits together represent a value. Perhaps I want to store a floating point in a byte with the following format:

$se_3e_2e_1e_0f_2f_1f_0$

where *s* is the sign, *e* is the biased exponent, and *f* is the fraction.

Let's say I have variable *r* that holds such a floating point number and I want to know the value of *e*:

(*r* & 0x78) >> 3

Alternatively I want to set *e* to 6:

(*r* & 0x87) | ((0x6 & 0xF) << 3)

The C Preprocessor

Lines that start with the symbol "#" are preprocessor directives. The preprocessor is an early stage in the compiling process.

```
#include <headerfile>
```

or

```
#include "headerfile"
```

Include a header file; use angled brackets to include standard library include files (e.g., #include <stdio.h> or #include <limits.h>; use double quotes to include your own header files (e.g., #include "tinycalc.h").

```
#define IDENTIFIER VALUE
```

Will cause all occurrences of IDENTIFIER to be replaced by VALUE. Can be used, for instance, to define constants in C programs. The "const" keyword is a relatively recent addition to the C language, before that #define was the only way to define constants.

```
#define IDENTIFIER(ARG1, ARG2, ...) «TOKEN STRING THAT MAY INCLUDE ARG1, ARG2, ...»
```

An extension of the previous definition where the identifier takes arguments. Will cause all occurrences of IDENTIFIER(ARG1, ARG2, ...) to be replaced with a version of the «TOKEN STRING» that has actual arguments substituted for formal parameters.