Miscellaneous

The `,` (Comma) Operator

The `,` (comma) operator is a binary operator. It first evaluates the its left operand, discards its value, and then evaluates its right operand. The value of the right operand is the value of the overall expression.

```
int x = 5;
while(--x, x >= 0) {
        // Do something
}
```

Command-line Arguments

There is an alternate allowable definition of main() in C and C++
`int main(int argc, char **argv)`
`argc` stands for argument count and stores the number of arguments in `argv`. `argv` stands for argument vector and is the array of arguments.
`argv[0]` is the name of the program. Hence, `argc` is one more than the number of command line arguments.
Note, the parameter name `argc` and `argv` are not mandatory, but are standard.

This is equivalent to the `echo` command

```
#include <stdio.h>

int main(int argc, char **argv) {
    for (argc--, argv++; argc > 0; argc--, argv++)
        printf("%s%c", *argv, (argc == 1) ? '\n' : ' ');
    return(0);
}
```

Bitwise Operations

Binary can be represented in C with a preceding `0b`.
Octal can be represented in C with a preceding `0`.

The `&` (bitwise and) operator performs the `&&` operation on each pair of corresponding bits of the operands.
`0b00001111 & 0b00111100` evaluates to `0b00001100`

The `|` (bitwise inclusive or) operator performs the `||` operation on each pair of corresponding bits of the operands.
`0b00001111 | 0b00111100` evaluates to `0b00111111`

The `^` (bitwise exclusive or) operator returns `0` if both operands are either `0` or `1` and returns `1` if exactly one operand is `1`. If the first operand is `1`, it returns the negation of the second operand. If the first operand is `0`, it returns the second operand without negation.
`0b00001111 ^ 0b00111100` evaluates to `0b00110011`

The `~` (bitwise complement) operator negates every bit in the operand.
`~0b00001111` evaluates to `0b11110000`

The `<<` (bitwise shift left) and `>>` (bitwise shift right) operators is a binary operator. The first operand is the value to shift and the second operand is the number of places to shift the value by. The `<<` and `>>` operators work like multiplication and integer division by `2`. The bitwise shift operators have lower precedence than the arithmetic operators.
`0b11001100 << 2` evaluates to `0b00110000`

The statement `(num |= (1 << k));` changes the `k` th bit of `num` to 1.
The statement `(num &= ~(1 << k));` changes the `k` th bit of `num` to 0.
The expression `(num & (1 << k))` checks if the `k` th bit of `num` is `1`.
The expression `(num ^= (1 << k))` toggles the `k` th bit of num.
The expression `(num = (num & ~(1 << k)) | (~num & (1 << k)))` toggles the `k` th bit of num.

Miscellaneous

`extern` can be used when declaring functions and variables in a file other than the one they are defined in.

Functions and variables can be declared multiple times however they can only be defined once in a scope.

The compiler only allocates memory when a variable is defined, not when it is declared.

Declaration `extern int i;`
Declaration, and definition `int i;`
Declaration, definition and initialisation `int i = 0;`

Variables declared in the body of a function belong exclusively to that function; they can't be examined or modified by other functions. In C89, variable declarations must come first, before all statements in the body of a function. In C99, variable declarations and statements can be mixed, as long as each variable is declared prior to the first statement that uses the variable.

In C99, the keyword `static` can be used to specify the minimum expected length of an array argument. This may help the compiler optimise the code better.

```
void function(int array[static 5]) {}
```

The compiler flag `-Werror=vla` can be used to generate an error for using variable length arrays.

Vim mapping `map <F8> :!gcc -Wall % -o %< && ./%< <CR>`

Use the `cdecl` program to understand casts and declarations.