We will be primarily concerned with the C compiler, `gcc`.

The program `gcc` is actually a front-end for a suite of programming tools.

For the purposes of CS 2505, the underlying tools include:

`cpp`	the GNU C preprocessor
`cc`	the GNU C language compiler
`as`	the GNU assembler
`ld`	the GNU linker

We will begin by considering only the use of `gcc` itself.

Download the example `caesar.c` from the course website if you want to follow along with the following examples.

Execute the following command: `gcc caesar.c`

You should not get any messages; list the files in your directory and you'll find a new file named `a.out` – that's the executable file produced by `gcc`.

Execute the command `a.out`; you should see a message from the program showing how to invoke it correctly.

Execute the command `a.out` with valid parameters, say:

```
a.out 3 AMansAManForAThat.txt
```

and you should see the modified file contents, echoed to the console window.

First of all, the default name for the executable file is `a.out`, which is both strange and unsatisfying.

Use the `-o` option to specify the name you want to be given to the executable:

```
gcc -o caesar caesar.c
```

```
1030 wmcquain in caesar> ls -l
total 8
-rw-rw-r--. 1 wmcquain comporg 1426 Sep 13 21:34 AMansAManForAThat.txt
-rw-rw-r--. 1 wmcquain comporg 2222 Sep 13 21:50 caesar.c
1031 wmcquain in caesar> gcc -o caesar caesar.c
1032 wmcquain in caesar> ls -l
total 20
-rw-rw-r--. 1 wmcquain comporg 1426 Sep 13 21:34 AMansAManForAThat.txt
-rwxrw-r--. 1 wmcquain comporg 9119 Sep 13 21:51 caesar
-rw-rw-r--. 1 wmcquain comporg 2222 Sep 13 21:50 caesar.c
1033 wmcquain in caesar>
```

Side note: as is often the case, the space after the `-o` option is optional.

Fundamental gcc options: -Wall

Use the -Wall option to direct gcc to display all relevant warning messages:

<div align="center">gcc -o caesar -Wall caesar.c</div>

```
1035 wmcquain in caesar> gcc -o caesar -Wall  caesar.c
caesar.c: In function 'processFile':
caesar.c:43:7: warning: implicit declaration of function 'isalpha' [-
Wimplicit-function-declaration]
      if ( isalpha(nextIn) ) {
      ^
caesar.c: In function 'checkShiftAmt':
caesar.c:86:8: warning: unused variable 'result' [-Wunused-variable]
    int result = (int) strtol(src, &p, 10);
        ^
1036 wmcquain in caesar>
```

So, the supplied C code *compiles*, but does not *compile cleanly*.

The first message tells us that at line 43 of the file caesar.c, in the function processFile, we have called a function isalpha that has not been declared explicitly.

All too true. The Standard Library function isalpha is declared in the header file ctype.h and the supplied code doesn't have an include directive for that header; in this case, we got away with that, but the issue should be fixed.

```
1035 wmcquain in caesar> gcc -o caesar -Wall  caesar.c
. . .
caesar.c: In function 'checkShiftAmt':
caesar.c:86:8: warning: unused variable 'result' [-Wunused-variable]
    int result = (int) strtol(src, &p, 10);
        ^
1036 wmcquain in caesar>
```

The next message tells us that at line 86 of the file `caesar.c`, in the function `checkShiftAmt`, we have declared a variable `result` whose value is never used.

Again, this is true.

However, in this case the variable was used in order to capture the return value from the library function `strtol`, which we do not make any further use of.

This is deliberate, and fits with the design of the function `checkShiftAmt`, and so we'll leave it unaltered.

You may also turn on certain specific categories of warnings if `-Wall` is too intrusive:

`-Wformat`

        warnings regarding mismatches between format specifiers and arguments

`-Wunused`

        warnings regarding unused variables

`-Wimplicit`

        warnings regarding functions that are called without being declared;
        usually results from missing `include` directives or misspellings

`-Wconversion`

        warnings regarding implicit conversions that could result in errors

`-Wshadow`

        warnings regarding name hiding

`-W`

        a variety of additional warnings not included in `-Wall`

```
1039 wmcquain in caesar> gcc -o caesar -Wall -W  caesar.c
caesar.c: In function 'applyShift':
caesar.c:57:36: warning: unused parameter 'shiftAmt' [-Wunused-parameter]
 char applyShift(char Original, int shiftAmt) {
. . .

1040 wmcquain@centosvm in ~/2505/notes/T05/caesar>
```

Now, we see an additional warning, and this one is somewhat alarming...

Why would a function receive a parameter and not use it? Sounds like a possible design error...

(In this case, the function `applyShift()` has not been completely implemented.

```c
// Adapted from An Introduction to GCC, Gough, p. 8
#include <stdio.h>

int main() {

    int a = 5, b = 10;
    printf("The sum of %d and %d is %f.\n", a, b, a + b);

    return 0;
}
```

```
1049 wmcquain in gough> gcc -o badformat -Wall badformat.c
badformat.c: In function 'main':
badformat.c:7:4: warning: format '%f' expects argument of type 'double',
but argument 4 has type 'int' [-Wformat=]
    printf("The sum of %d and %d is %f.\n", a, b, a + b);
    ^
1050 wmcquain in gough> badformat
The sum of 5 and 10 is 0.000000.
```

Use the `-std` option to direct `gcc` to require code to comply with a particular C langauge standard:

```
gcc -o caesar -Wall -std=c11 caesar.c
```

This is necessary in order to use some features added in C11 (such as declaring for-loop counters within the loop header).

It is also necessary with some legacy code in order to sidestep requirements imposed by newer standards.

Unless explicitly stated otherwise, in CS 2505 we will always specify compliance with the C99 standard (as shown above).

The official reference manual for GCC is available at:

http://gcc.gnu.org/onlinedocs/

This is a very useful, if somewhat verbose, resource.

Executing `gcc` with the `-save-temps` option results in the preservation of some temporary intermediate files created by/for the underlying tools.

For example if you use the file `caesar.c`:

`caesar.i`	written by the preprocessor; input to the compiler
`caesar.s`	written by the compiler; input to the assembler
`caesar.o`	written by the assembler; input to the linker
`caesar`	written by the linker

By default, only the final, executable file is preserved once the process is complete.

We will gradually see that the intermediate files are occasionally of use, if for no reason than that they shed light on the actual process of program translation from a high-level language to the machine level.

Try executing the command

<div align="center">

`cpp caesar.c > caesar.i`

</div>

`cpp` writes its output to standard output; this *redirects* it into a (new) file named `caesar.i`.

If you examine this (text) file, the first 2000 or so lines indicate the processing of the `include` directives in the source file; so declarations from those files are available to the compiler.

At the end of the file, you will find a modified copy of the original source:

-    all the comments have been stripped out

-    the values that were `defined` in the source file have been substituted into the source code

Now, try executing the command

```
cc –S caesar.i
```

With the –S option (case-sensitive!), the compiler writes its output to a file; the name is generated from the name of the input file; in this case, the output is written to caesar.s.

This file contains the assembly code generated by the compiler from the pre-processed C source code.

Now, try executing the command

```
as -o caesar.o caesar.s
```

The assembler writes its output to `a.out` by default; the name can be specified using the option `-o`, as with `gcc`.

This file contains a partial translation of the assembly code into native machine language code; calls to library functions haven't been resolved completely since the code for those is not in the local source file.

Now, try executing the command

```
ld caesar.o
```

The linker will complain about a large number of undefined references because it doesn't know where to find the system files that contain the implementations of the relevant features in the Standard Library.

Fortunately, gcc takes care of these settings for us and invokes the linker as needed:

```
gcc -o caesar caesar.o
```

More useful gcc options

You may also turn on certain specific categories of warnings if `-Wall` is too intrusive:

`-lm`

        required in order to use floating-point functions in the Library

`-std=c11`

        required to use features specific to the C11 Standard
        other options include: `c89, c99`

`-On`

        choose optimization level n (0, 1, 2 or 3); use `-O0` for debugging

`-m32, -m64`

        build 32-bit or 64-bit executable (we use only 64-bit code here)

`-g, -ggdb, -ggdb3`

        generate debugging information to aid gdb