C Program Structure

C Preprocessor

- File inclusion. The command is `#include`. Files in the standard library can be specified using angle brackets. Files specified in double quotes are relative to the current directory. For example:

```
#include <stdio.h>
#include "myheader.h"
```

- Macro substitution. The command is `#define`. Macros expand into in-line code. There is no compile-time check for type correctness. For example:

```
#define max(a,b) ((a)>(b)? (a) : (b))
```

- Defining constants

```
#define MAX 323239283
```

Conditional Inclusion

```
#ifdef __linux__
    printf("I am the Happy Penguin!\n");
#elif _WIN32
    printf("Welcome to MS Windows ( I rule!).\n");
#elif __APPLE__&&__MACH__
    printf("Welcome to I am cool!\n");
#else
    printf("Uh! who am i?\n");
#endif
```

Note that the above ifdef strings are standard ways of detecting the Operating System the code is being compiled on.
See http://sourceforge.net/p/predef/wiki/OperatingSystems/ for full list of identifying strings for various Operating Systems.

```c
/* C-examples/c-preprocessor/ex1.c */
#include <stdio.h>
#include <stdlib.h>
#define DEBUG 1
const int IN=1;  /* inside a word */
const int OUT=0; /* outside a word */
/* count number of characters, words and lines in the standard input */
int main(int argc, char *argv[]) {
    char c;
    long nc, nw, nl;
    int state;
    state = OUT;
    nl = nw = nc = 0;
    while ((c = getchar()) != EOF ) {
        nc++;
#ifdef DEBUG
        printf("nc=%d\n",nc);
#endif
        if (c == '\n') {
            nl++;
#ifdef DEBUG
            printf("nl=%d\n",nl);
#endif
        }
        if (c == ' ' || c == '\n' || c == '\t')
            state = OUT;
        else if (state == OUT) {
                state = IN;
                nw++;
#ifdef DEBUG
                printf("nw=%d\n",nw);
#endif
        }
    }
    printf("%ld %ld %ld\n", nl, nw, nc);
    exit(0);
}
```

Scope

- Variables and functions are visible from the point they are defined until the end of the source file.
- Creating function prototypes at the start of the source file (or in an included header file) makes them visible throughout the source file.
- Add modifier `static` in front of a function declaration makes the scope of the function to be limited to the containing source file (similar to `private` modifier in Java).

Scope (2)

- Global variables are not visible to code in another source file. To make a global variable be visible to multiple source files, define it normally in only one source file and then declare it as `extern` in all the other source files.
- The *declaration* of an external variable announces the properties of the variable (e.g. type, name).
- The *definition* of an external variable will actually *create and set aside storage for the variable*.
- There must be only one *definition* of an external variable among *all* files that make up the source program.

```
/* file1.c */      /* file2.c */        /* file3.c */
int moomoo = 10;   extern int moomoo;   extern int moomoo;
```

Example

- Discuss examples `ourhdr.h`, `scope1.c` and `scope2.c`.

ourhdr.h

```
/* C-examples/scope/ourhdr.h */
#ifndef __OURHDR_H
#define __OURHDR_H

/* function prototypes */
int f1(int, double);
void f2(int);
void f3(void);

extern int total;

#endif /* __OURHDR_H */
```

scope1.c

```
/* C-examples/scope/scope1.c */
#include <stdio.h>
#include <stdlib.h>
#include "ourhdr.h"
int total = 1;
static int f4(int); /* private function */

int f1(int x, double z) {
    static int counter=0;
    counter++;
    printf("counter in f1() = %d\n", counter);
    x += f4((int) z);
    return (x*z);
}
static int f4(int x) {
    return 2*x;
}
int subtotal = 0;

void f3() {
    total += 10;
    subtotal = total;
    if (total < 100)
        total = f1(total, 1.5);
}
```

scope1.c (contd.)

```
int main (int argc, char **argv) {
    int m;
    double z;
    int x;

    if (argc != 3) {
        fprintf(stderr,"Usage: %s <int> <double> \n",argv[0]);
        exit(1); /* return unsuccessful status to the shell */
    }
    m = atoi(argv[1]);
    z = atof(argv[2]);

    x = f1(m, z);
    f2(m);
    f3();
    x = x + f1(m,z);

    printf("total = %d\n",total);
    printf("subtotal = %d\n",subtotal);
    return 0;
}
```

scope2.c

```
/* C-examples/scope/scope2.c */
#include <stdio.h>
#include <stdlib.h>
#include "ourhdr.h"

void f2(int x)
{
    total += x;
}
```

Convention for writing header files

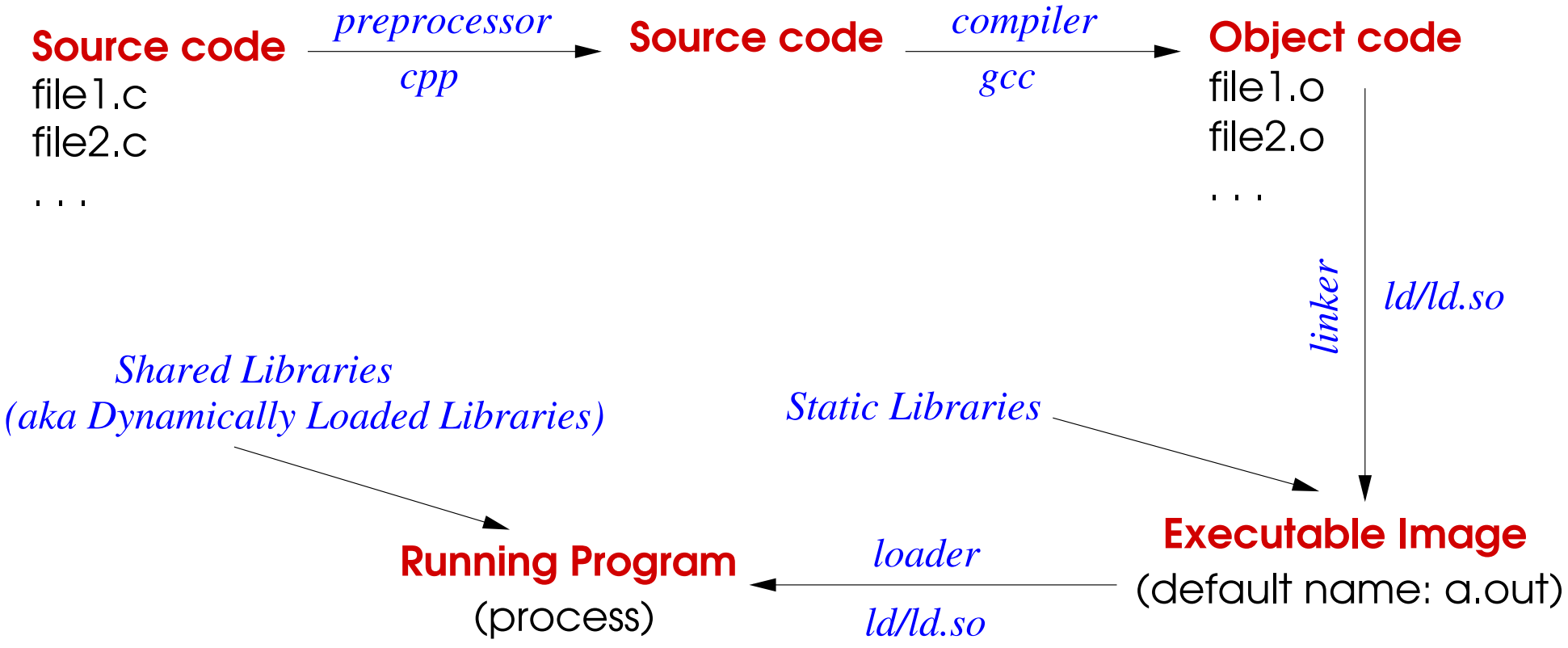
- Note that by using the ifdef's to wrap the header file is a standard way of declaring header files. This prevents the contents of the header file from being included multiple times (which would cause compilation errors)
- The convention is to use two leading underscores, convert the letters in the name of the header file to all uppercase and convert periods to underscores. You should always follow this convention.

```
/* C-examples/scope/ourhdr.h */
#ifndef __OURHDR_H
#define __OURHDR_H

/* declarations */

#endif /* __OURHDR_H */
```

Compiling, Linking and Running Programs



Note: The above illustrates the GCC compiler tool chain but the concepts are the same for any C/C++ compiler.

The GNU C/C++ compiler options

An integrated C/C++ open-source compiler. Available on a wide variety of platforms. The following shows some commonly used command line. See the man page for more options.

- `-c` Compile but do not link.
- `-Wall` Print all warnings. *Always use this option...some of the warnings can save you hours of debugging!*
- `-O` Optimize option. Usually speeds up the execution time significantly.
- `-g` Enable debugging. Need this option if you want to use a debugger. Also gcc allows you to use the optimize option (-O) at the same time as the debug option.
- `-o <filename>` Name the output executable file to the given filename. Otherwise the default file name is `a.out`.
- `-I dir` Add the directory `dir` to the list of directories to be searched for header files. Directories named by `-I` are searched before the standard system include directories.

The GNU C/C++ compiler options

- `-D<macro>=[<name>]` Define a macro. The C file is processed with the macro defined. For example `gcc -Wall -DDEBUG ex1.c` would define the DEBUG macro to be true for the file being compiled.
- `-E` Stop after the preprocessing stage; do not run the compiler proper. The output is in the form of preprocessed source code, which is sent to the standard output.
- `-S` Stop after the stage of compilation proper; do not assemble. The output is in the form of an assembler code file for each non-assembler input file specified.
- `-version` Display the version number and copyrights of the invoked GCC

Recommended Assignment

- Read Sections 4.4 through 4.11 of the C book.
- Attempt Exercises 4-3, 4-12, 4-13 and 4-14.