C Tutorial - Program Organization

CS 537 - Introduction to Operating Systems

Simple Compilation
- To compile a program in Unix, use gcc:
- Example:
  - `prompt> gcc myProgram`
- The previous example compiles an executable file called `a.out`
- To give the executable a different name, use the `-o` option
- Another Example
  - `prompt> gcc myProgram -o myProg`
- There are lots more command line options
  - see the man pages

File Organization
- Sophisticated programs have multiple files
  - program files (*.c)
    - most of your code goes into these files
  - header files (*.h)
    - mostly prototypes and structure definitions
  - object files
    - compiled program files that are not linked
    - must to be linked to become executable files
  - make file
    - always named Makefile
    - responsible for building the entire program
Program Files

- This is what gets compiled
  - includes all of the function definitions
  - before a function can be used, it must be declared
    - prototypes
    - these can either be declared in the program file or in
      a header file
- Always followed with a .c extension

Program File Dependencies

- Possible for a function defined in one program file to be used in another
  - the two (or more) program files must be linked
- The simple gcc compilation example shown earlier automatically links your program to some standard libraries
  - that is why you don't need to define scanf in your code
- If you want to link to non-standard libraries, you must include them in the compilation command
  - `gcc myProg.c common.c -o myProg`
    - this compiles both myProg.c and common.c and links them together

Example

```c
#include <stdio.h>
#include "common.h"
int main() {
    int x, y;
    x = getNumber();
    y = getNumber();
    printf("x = %d, y = %d\n", x, y);
    return 0;
}
```
```c
common
void getNumber(int *num) {
    printf("Number: ");
    scanf("%d", num);
}
void printChar(int ch, int n) {
    printf("%c %d\n", ch, n);
}
void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}
```
Header Files

- Often put all the prototypes for a specific program file into a separate file
- call this the header file
- always ends in a .h extension
- Helps the program files look cleaner
- Allows prototypes of one program file to be easily included in another program file
- remember, can’t use a function until declaring it
- Example

```c

```

Object Files

- Instead of compiling multiple files together, they can be compiled separately into object files
- object files are not executable
- multiple object files can then be linked to form an executable
- use the -c option of gcc to create object files
- Example
- `prompt> gcc myProg.c -c`
- `prompt> gcc common.c -c`
- these two lines create `myProg.o` and `common.o` object files
- `prompt> gcc myProg.o common.o -o myProg`
- linked `myProg.o` and `common.o` to create the executable `myProg`

```
```

gcc

- The gcc program is both a compiler and a linker
- it can compile *.c files into object files using the -c option
- it can link multiple object files (*.o) and create an executable
- `-o` output file name
- `-o` output file name
- it can both compile and link if multiple *.c files are given on the same line
make

- Obviously, for very large programs (with many files), creating and linking all of the object files could be very tedious
- This process can be automated using make
- Makefile
  - this is a file that defines exactly how a program should be compiled and linked
  - it only compiles what needs compiling
    - if a file has not been modified since it was last compiled, it won't recompile it
- Whenever you run make, it finds the Makefile file
  - it then executes all the operations defined in Makefile

Makefile

- The basic make file consists of targets, rules, and commands
  - target: name of the object to be built
  - rules: which files, if modified, would require this target to be re-built
  - commands: how to rebuild a target if any of objects listed in the rules have been changed

Example

```
src
Makefile
  rule
  myProg
  gcc myProg common-o myProg
  myProg
  gcc myProg -c
  common
  gcc common.c
  command
```
More on Makefile

- To help simplify and generalize your makefile, variables can be used
  - They are usually declared as all caps
  - They are proceeded by a dollar sign ($) and enclosed in parenthesis when used
- Comments can be included
  - the line must start with a pound sign (#)
  - entire line is ignored

Example Redone

```
Makefile
# compiler
CC=gcc
# linker
LD=gcc
# object files
OBJS=CObj0 common.o
mpProj: $OBJS
      cc $OBJS -o mpProj
mpProj mpProj
      cc common.o -o mpProj
common common common.h
      cc common common.c
```

More on make

- There are million and one options for make
- There are an equal number of uses
- Find a good book or web page to learn more
  - what you have seen here should get you started