Sample Code

Let's say you want to write a program that takes at the command-line one integer input, and determines whether that integer is a prime number or not. Now let's say that you don't want to write your own code for determining primality, so you ask your friend, who you know has written such a function already. She sends you a pair of files (prime.h and prime.c). Her header file (prime.h) looks like this:

```
int isPrime(int n); // returns 0 if n is not prime, 1 if n is prime
```

So we know a couple of things from this header file. It declares a function prototype for isPrime(). We can see this function takes a single integer as an input argument, and returns an integer value: 1 if the input value is a prime number, and 0 if it is not. Now we know all we need to know in order to use this function (without even looking at the function's source code, which resides in primes.c).

Here is what the prime.c file look like:

```
int isPrime(int n) {

  // returns 0 if not prime, 1 if prime

  if (n<2) return 0;         // first prime number is 2
  if (n==2) return 1;        // ensure 2 is identified as a prime
  if ((n % 2)==0) return 0;  // all even numbers above 2 are not prime

  int i;
  for (i=3; i*i < n; i++) {  // test divisibility up to sqrt(n)
    if ((n % i) == 0) {
      return 0;
    }
  }
  return 1;
}
```

Here is what our program driver.c looks like:

```
#include <stdio.h>
#include <stdlib.h>
#include "primes.h"

int main(int argc, char *argv[]) {
  if (argc < 2) {
    printf("error: must provide a single integer value to test\n");
    return 1;
  }
  else {
    int n = atoi(argv[1]);
    int prime = isPrime(n);
    printf("isPrime(%d) = %d\n", n, prime);
    return 0;
  }
}
```

The makefile looks like:

```
executableName = isPrimeCheck.out
driver = driver

# Makes
all: $(driver).o prime.o
	gcc $(driver).o prime.o -o $(executableName)

$(driver).o: $(driver).c
	gcc -c $(driver).c

prime.o: prime.c
	gcc -c prime.c

# To clean the compilation.
clean:
	rm -f *.o $(executableName)
```

Run the following commands in the terminal:

```
make

./isPrimeCheck.out <number>
```

Understanding the Makefile

```
executableName = isPrimeCheck.out
driver = driver
```

executableName: This variable holds the name of the final executable file
driver: This variable holds the base name of the driver source file (driver). The .c extension will be added later in the rules.

```
# Makes
all: $(driver).o prime.o
	gcc $(driver).o prime.o -o $(executableName)
```

all: This is the default target that is executed when you run make without any arguments. It depends on the object files driver.o and prime.o.
gcc $(driver).o prime.o -o $(executableName): This command links the object files driver.o and prime.o to create the executable isPrimeCheck.out.

```
$(driver).o: $(driver).c
	gcc -c $(driver).c
```

$(driver).o: This target is for compiling the driver.c source file into an object file driver.o.
gcc -c $(driver).c: This command compiles driver.c without linking, producing the object file driver.o.

```
prime.o: prime.c
	gcc -c prime.c
```

prime.o: This target is for compiling the prime.c source file into an object file prime.o.
gcc -c prime.c: This command compiles prime.c without linking, producing the object file prime.o.

```
# To clean the compilation.
clean:
	rm -f *.o $(executableName)
```

clean: This target is used to remove the compiled object files and the executable. It is not run automatically and must be invoked explicitly by running make clean.
rm -f *.o $(executableName): This command removes all object files (*.o) and the executable (isPrimeCheck.out). The -f flag forces the removal without prompting for confirmation and suppresses errors if the files do not exist.