1). This `f.c` program will print out all the factors for all the given positive integers in the `p x q` format:

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

static void         printFactors( int );

extern int
main( int argc, char *argv[] )
{
     int       i;
     int       n = argc - 1;

     for ( i = 1; i <= n; i++ )
     {
          if ( i > 1 )
          {
               printf( "\n" );
          }
          printFactors( atoi( argv[ i ] ) );
     }

     return 0;
}

static void
printFactors( int n )
{
     int       i = 1;
     int       f = n;

     printf( "%d:\n", n );
     do
     {
          if ( ( n % i ) == 0 )
          {
               f = n / i;
               printf( "%d x %d\n", i, f );
          }
          i++;
     }
     while ( i < f );
}
```

Save the above code in the `f.c` file.

Build it by typing the following at the terminal:

```
gcc -g -o f f.c
```

Run it:

```
./f 60 120
```

In this case we gave the factorization program two positive integers: `60` and `120`. The output is:

```
60:
1 x 60
2 x 30
3 x 20
4 x 15
5 x 12
6 x 10

120:
1 x 120
2 x 60
3 x 40
4 x 30
5 x 24
6 x 20
8 x 15
10 x 12
```

2). This `wcgr.c` program will print out all the Whole Clock Gear Ratios for two given factors. The lowest tooth count is `5`, the highest is `150`:

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

static void             printRatios( int );

extern int
main( int argc, char *argv[] )
```

```
{
        int             i;
        int             n = argc - 1;

        for ( i = 1; i <= n; i++ )
        {
                if ( i > 1 )
                {
                        printf( "\n" );
                }
                printRatios( atoi( argv[ i ] ) );
        }

        return 0;
}

static const int        L = 5;
static const int        H = 150;

static void
printRatios( int f )
{
        int             p;
        int             q;
        int             n = 0;
        int             N = H / f;

        printf( "%d:1 ratios:\n", f );
        for ( q = L; q <= N; q++ )
        {
                p = f * q;
                if ( n == 10 )
                {
                        n = 0;
                        printf( "\n" );
                }
                if ( n )
                {
                        printf( " " );
                }
                n++;
                printf( "%d:%d", p, q );
        }

        if ( n < 10 )
        {
                printf( "\n" );
```

```
        }
}
```

Save it in the `wcgr.c` file.

Build it by typing the following at the terminal:

```
gcc -g -o wcgr wcgr.c
```

Run it:

```
./wcgr 5 12
```

In this case we gave the whole ratios printer program two positive integers: `5` and `12` which multiply to `60`. The output is:

```
5:1 ratios:
25:5 30:6 35:7 40:8 45:9 50:10 55:11 60:12 65:13 70:14
75:15 80:16 85:17 90:18 95:19 100:20 105:21 110:22 115:23 120:24
125:25 130:26 135:27 140:28 145:29 150:30

12:1 ratios:
60:5 72:6 84:7 96:8 108:9 120:10 132:11 144:12
```

3). This `nwcgr.c` program will print out all the Non-Whole Clock Gear Ratios for two given factors which will also include the relatively prime ratios. The lowest tooth count is `5`, the highest is `150`:

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

static void             genRatios( int, int );
static void             checkRatio( int, int, int, int );

extern int
main( int argc, char *argv[] )
{
        int             f1 = atoi( argv[ 1 ] );
        int             f2 = atoi( argv[ 2 ] );

        genRatios( f1, f2 );

        return 0;
}

static const int        L = 5;
```

```
static const int        H = 150;

static void
genRatios( int f1, int f2 )
{
        int             p;
        int             q;
        int             N = H / f1;

        for ( q = L; q <= N; q++ )
        {
                p = f1 * q;
                checkRatio( f1, p, q, f2 );
        }
}

static void
checkRatio( int f1, int p, int q, int f2 )
{
        int             m;
        int             n;
        int             N = H / f2;
        int             f = f1 * f2;

        for ( n = L; n <= N; n++ )
        {
                m = f2 * n;
                if ( ( ( p % n ) != 0 ) && ( ( m % q ) != 0 ) )
                {
                        printf( "%d:%d %d:%d\n", p, n, m, q );
                }
        }
}
```

Save it in the `nwcgr.c` file.

Built it by typing the following at the terminal:

```
gcc -g -o nwcgr nwcgr.c
```

Run it:

```
./nwcgr 5 12
```

In this case we gave the non-whole ratios printer program two positive integers: `5` and `12` which multiply to `60`. A small portion of the whole output is:

```
25:6 72:5
25:7 84:5
25:8 96:5
25:9 108:5
25:11 132:5
25:12 144:5
35:6 72:7
35:8 96:7
35:9 108:7
35:10 120:7
35:11 132:7
35:12 144:7
```

4). This `rpcgr.c` program will print out all the Relatively Prime Clock Gear Ratios for two given factors. The lowest tooth count is `5`, the highest is `150`:

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

static void             genRatios( int, int );
static void             checkRatio( int, int, int, int );
static int              gcd( int, int );

extern int
main( int argc, char *argv[] )
{
        int             f1 = atoi( argv[ 1 ] );
        int             f2 = atoi( argv[ 2 ] );

        genRatios( f1, f2 );

        return 0;
}

static const int        L = 5;
static const int        H = 150;

static void
genRatios( int f1, int f2 )
{
        int             p;
        int             q;
```

```
        int             N = H / f1;

        for ( q = L; q <= N; q++ )
        {
                p = f1 * q;
                checkRatio( f1, p, q, f2 );
        }
}

static void
checkRatio( int f1, int p, int q, int f2 )
{
        int             m;
        int             n;
        int             N = H / f2;
        int             f = f1 * f2;

        for ( n = L; n <= N; n++ )
        {
                m = f2 * n;
                if ( gcd( p, n ) == 1 && gcd( m, q ) == 1 )
                {
                        printf( "%d:%d %d:%d\n", p, n, m, q );
                }
        }
}

static int
gcd( int n1, int n2 )
{
        int             m;

        while ( n2 != 0 )
        {
                m = n1 % n2;
                n1 = n2;
                n2 = m;
        }

        return n1;
}
```

Save it in the `rpcgr.c` file.

Build it by typing the following at the terminal:

```
gcc -g -o rpcgr rpcgr.c
```

Run it:

```
./rpcgr 5 12
```

In this case we gave the relatively prime ratios printer program two positive integers: `5` and `12` which multiply to `60`. A small portion of the whole output is:

```
25:11 132:5
25:12 144:5
35:6 72:7
35:8 96:7
35:9 108:7
35:11 132:7
35:12 144:7
55:6 72:11
55:7 84:11
55:8 96:11
55:9 108:11
55:12 144:11
65:6 72:13
65:7 84:13
```

5). This `gcd.c` program will print out the Greatest Common Divisor of two positive integers:

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

extern int
main( int argc, char *argv[] )
{
        int             m;
        int             n1 = atoi( argv[ 1 ] );
        int             n2 = atoi( argv[ 2 ] );


        printf( "gcd(%d, %d) = ", n1, n2 );

        while ( n2 != 0 )
        {
                m = n1 % n2;
                n1 = n2;
```

```
                n2 = m;
        }

        printf( "%d\n", n1 );
}
```

Save it in the `gcd.c` file.

Build it by typing the following at the terminal:

```
gcc -g -o gcd gcd.c
```

Run it:

```
./gcd 5 3
```

In this case we gave the `gcd` program two positive integers: `5` and `3`. The output is:

```
gcd(5, 3) = 1
```

6). This `lcm.c` program will print out the Least Common Multiple of two positive integers:

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

extern int
main( int argc, char *argv[] )
{
        int             m;
        int             n1 = atoi( argv[ 1 ] );
        int             n2 = atoi( argv[ 2 ] );
        int             lcm = n1 * n2;


        printf( "lcm(%d, %d) = ", n1, n2 );

        while ( n2 != 0 )
        {
                m = n1 % n2;
                n1 = n2;
                n2 = m;
        }

        lcm /= n1;

        printf( "%d\n", lcm );
}
```

Save it in the `lcm.c` file.

Build it by typing the following at the terminal:

```
gcc -g -o lcm lcm.c
```

Run it:

```
./lcm 5 3
```

In this case we gave the `lcm` program two positive integers: `5` and `3`. The output is:

```
lcm(5, 3) = 15
```