12 Command-line arguments

To be able to supply your program with command-line arguments, add a bit to your main() section. First, more notation about characters and character strings.

- A single character (as opposed to a 'string' of characters) is represented with a single quote, such as

  `'a', 'A', '\n', '\0'`

- The **null** character is represented as '\0', 8 zero-bits or `00000000`

- A character *string* is an array of characters, terminated with a null character. For example,

  `"hello\n"`

  is stored as an array of *seven* characters, including the final null character.

- For technical reasons,

  `char * x;`

  declares x as a character string.

In

```
a.out 1 2 3
```

the machine-code program a.out is called with *command-line arguments* 1 2 3 respectively.

For example, the euclidean program can be made useful as follows

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

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

   ... add euclid code here
}
```

The variable `argv` is an *array* of *character strings*. Its size is not given, but `argv[i]` is the i-th command argument, valid for i between 0 and `argc-1`.

The command-line arguments are character strings, but they can be converted to integers, etcetera, through another `#include`:

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

12.1 If statements

```
  if ( condition )
  {  do_this; }
 [ optionally
  else
  {  do_that; }
 ]
```

The first application is checking command-line arguments, e.g.,

```
main ( int argc, char * argv[] )
{
  if ( argc != 3 )
  {
    fprintf (stderr,"usage %s m n\n", argv[0]);
    exit(-1); // Unusual exit from program.
  }

  int m = atoi ( argv[1] ),
    n = atoi ( argv[2] );

  if ( m <= 0 || n <= 0 )
  {
    fprintf(stderr,"args must be positive\n"0;
    exit (-1);
  }

  ... continue with 'euclid' code here ...
}
```

12.2 Nested if-statements

The if-statement is not associative, so to speak. For example, if one is parsimonious with curly braces, one might write

```
  if ( A )      Which part is governed by 'if A?'
    if ( B )    The indentation suggests: lines 2,3
      C;        Wrongly.  The else balances the 'nearest
  else          if.'
    D;          It should be written as

  if ( A )    equivalent to   if ( A )
    if ( B )                  { if ( B )
      C;                          C;
    else                        else
```

```
    D;                          D;
                              }
```

A ‘cascading set of if statements’ might be indented as shown

```
  if ( A )                NO:    if ( A )
    { statements 1 }               { statements 1 }
  else if ( B )                    else if ( B )
       { statements 2 }            { statements 2 }
       else if ( C )               else if ( C )
            { statements 2 }       { statements 2 }
```

```
This occurs often, and indentation would push the code
off the page.
```

12.3 Conditions are integers

In C, any integer value can appear in a condition, and every conditional expression will be converted to integer (0 or 1, ‘boolean.’) For example,

```
main( int argc, char argv[] )
{
  int ok = 1;

  if ( argc != 3 )
    ok = 0;
  else
  {
    m = atoi ( argv[1] );
    n = atoi ( argv[2] );
    if ( m <= 0 )
      ok = 0;
    if ( n <= 0 )
      ok = 0;
  }

  if (!ok)
  {
    fprintf( stderr, "Usage %s m n, (m,n positive)\n", argv[0]);
    exit(-1);
  }
}
```

Or, supposing that `dd,mm,yy` give day month and year (year 1582 onwards).

```
  int leap_year =
    yy % 4 == 0 && ( yy % 100 != 0 || yy % 400 == 0);
```