1. Comparing text files. This is an entirely trivial program, that tests whether two text files are equal, and if not so, points out the first point of difference.

```
#include <stdio.h>
#include <stdlib.h>
  typedef char bool;
```

2. The outline of the program is simple. We read characters from both input files into c_1 and c_2 until the comparison is complete. Line and column counts are maintained in *line* and *col*.

```
#define left_margin 1    /* leftmost column number; change to 0 if you prefer */
  ⟨ Functions 5 ⟩
  int main (int n, char **arg)
  { FILE *f₁, *f₂;    /* the two input files */
    int c₁, c₂, col ⇐ left_margin;
    long line ⇐ 1;
    ⟨ Open the files f₁ and f₂, taking their names from the command line or from the terminal; in case of
        an error for which no recovery is possible, call exit(1) 6 ⟩
    ⟨ Search for first difference, leaving c₁ ≠ c₂ if and only if a difference was found 3 ⟩
    ⟨ Report the outcome of the comparison 4 ⟩
    return 0;    /* successful completion */
  }
```

3. The heart of the program is this simple loop. When we reach the end of one of the files, the files match if and only if the other file has also reached its end. For this reason the test $c_1 = c_2$, which requires characters to be read from both files, must precede the test for file end; when only one file ends, it is the former test which breaks the loop.

```
⟨ Search for first difference, leaving c₁ ≠ c₂ if and only if a difference was found 3 ⟩ ≡
  while ((c₁ ⇐ getc(f₁)) = (c₂ ⇐ getc(f₂)) ∧ c₁ ≠ EOF)
    if (c₁ = '\n') { ++line; col ⇐ left_margin; } else ++col;
```

This code is used in section 2.

4. When the first difference occurs at the end of one of the files, or at the end of a line, we give a message indicating this fact.

```
⟨ Report the outcome of the comparison 4 ⟩ ≡
  if (c₁ = c₂) printf("Files␣match.\n");
  else
  { printf("Files␣differ.\n");
    if (c₁ = EOF ∨ c₂ = EOF)
      { the_file(c₁ = EOF); printf("is␣contained␣in␣the␣other␣as␣initial␣segment.\n"); }
    else if (c₁ = '\n' ∨ c₂ = '\n')
      { the_file(c₁ = '\n'); printf("has␣a␣shorter␣line␣number␣%ld␣than␣the␣other.\n", line); }
    else printf("First␣difference␣at␣line␣%ld,␣column␣%d.\n", line, col);
  }
```

This code is used in section 2.

5. The function *the_file* starts a sentence about the first or second file, depending on its boolean argument.

```
⟨ Functions 5 ⟩ ≡
  void the_file (bool is_first) { printf("The␣%s␣file␣", is_first ? "first" : "second"); }
```

See also section 7.

This code is used in section 2.

6. There can be be zero, one or two command line arguments. If there are none, the user is prompted to supply them, and if there are two these are taken as the file names, prompting the user only in case a file could not be opened. In case just one argument is present, the first file is assumed to be the standard input, which does not have to be opened; in this case however we will not read a file name from terminal in case the second file cannot be opened.

#define *read_mode* "r"

⟨ Open the files f_1 and f_2, taking their names from the command line or from the terminal; in case of an error for which no recovery is possible, call *exit*(1) 6 ⟩ ≡

```
--n; ++arg;     /* ignore "argument" 0, which is the program name */
if (n = 0)
  { open_file(&f1, "First␣file␣to␣compare", ⊙); open_file(&f2, "Second␣file␣to␣compare", ⊙); }
else if (n = 1)
{ f1 ⇐ stdin;
  if ((f2 ⇐ fopen(*arg, read_mode)) = ⊙) { printf("Could␣not␣open␣file␣%s.\n", *arg); exit(1); }
}
else if (n = 2)
{ open_file(&f1, "Give␣another␣first␣file", *arg++);
  open_file(&f2, "Give␣another␣second␣file", *arg);
}
else { printf("No␣more␣than␣two␣command␣line␣arguments␣are␣allowed.\n"); exit(1); }
```

This code is used in section 2.

7. The function *open_file* will try to open the file *name* for reading, and if this fails it will prompt for another file name until it has success. If called with *name* = ⊙, the function starts with prompting right away.

⟨ Functions 5 ⟩ +≡

```
void open_file (FILE **f, char *prompt, char *name)
{ char buf[80];
  if (name = ⊙ ∨ (*f ⇐ fopen(name, read_mode)) = ⊙)
    do { printf("%s:␣", prompt); fflush(stdout); scanf("%79s", buf); }
    while ((*f ⇐ fopen(buf, read_mode)) = ⊙);
}
```

8. Index.

arg: 2, 6.
bool: 1, 5.
buf: 7.
col: 2, 3, 4.
c_1: 2, 3, 4.
c_2: 2, 3, 4.
EOF: 3, 4.
exit: 6.
f: 7.
fflush: 7.
fopen: 6, 7.
f_1: 2, 3, 6.
f_2: 2, 3, 6.
getc: 3.
is_first: 5.
left_margin: 2, 3.
line: 2, 3, 4.
main: 2.
n: 2.
name: 7.
open_file: 6, 7.
printf: 4, 5, 6, 7.
prompt: 7.
read_mode: 6, 7.
scanf: 7.
stdin: 6.
stdout: 7.
the_file: 4, 5.