1. Intro. A `CWEB` program that does permutation multiplication in the cycle notation. To facilitate testing the input of this program will look like `(acfg)(bcd)(aed)(fade)(bgfae)`. In general, the alphabet is supposed to be (a subset of) `a--z`. The output for this case should be `(adg)(bce)`. The 'leaders' are increasing; the cycles start with the smallest letter. This is basically a solution to exercise 1.3.3-8 in *Fundamental Algorithms*.

```
#include <stdio.h>
  int iperm[256];    /∗ the inverse of the permutation read so far ∗/
  int perm[256];    /∗ the permutation ∗/
  int main()
  {
    ⟨Read input and construct iperm 2⟩;
  out:
    for (int i = 0; i < 256; ++i) perm[iperm[i]] = i;
    ⟨Produce output from perm 4⟩;
  }
```

2. We have *iperm*[`'a'`] ≡ `'c'` if `c` should change into `a`. When we see `xy` in the input that means that the letter which we thought so far that will change into `x` will actually change into `y`.

```
⟨Read input and construct iperm 2⟩ ≡
  int start;    /∗ the start of the cycle ∗/
  int prev;    /∗ what becomes the previously read character ∗/
  int current;    /∗ the current character ∗/
  int t;    /∗ temporary variable for swapping ∗/
  for (current = 0; current < 256; ++current) iperm[current] = current;
  ⟨Get next character in current 3⟩;
  while (1) {
    while (current ≡ '(') {
      ⟨Get next character in current 3⟩;
      start = current;
    }
    if (current ≡ ')') current = start;
    t = iperm[current]; iperm[current] = prev; prev = t;
    ⟨Get next character in current 3⟩;
  }
```

This code is used in section 1.

3. ⟨Get next character in *current* 3⟩ ≡

```
  do current = getchar();
  while (¬((current ≥ 'a' ∧ current ≤ 'z') ∨ current ≡ ')' ∨ current ≡ '(' ∨ current ≡ EOF));
  if (current ≡ EOF) goto out;
```

This code is used in section 2.

4. One-element cycles are not printed.

⟨ Produce output from *perm* 4 ⟩ ≡

```
while (1) {
  for (start = 0; start < 256 ∧ perm[start] ≡ start; ++start) ;
  if (start ≡ 256) break;
  printf("(%c", start);
  for (current = perm[start]; current ≠ start; current = perm[current]) {
    perm[iperm[current]] = iperm[current];
    printf("%c", current);
  }
  perm[iperm[start]] = iperm[start];
  printf(")");
}
printf("\n");
```

This code is used in section 1.

5. Index.

$\langle$ Get next character in *current* 3 $\rangle$ Used in section 2.
$\langle$ Produce output from *perm* 4 $\rangle$ Used in section 1.
$\langle$ Read input and construct *iperm* 2 $\rangle$ Used in section 1.

PERM