1. Intro. Computes the inverse of a permutation in place using algorithm J from TAoCP. This is also a solution to exercise 1.3.3-13 that asks for a proof of correctness.

2. Here is how the algorithm finds the inverse permutation of 024135 (here $\overline{x} = -x - 1$):

$$\overline{024135} \to^5 \overline{02413}5 \to^4 \overline{024}4\overline{1}5 \to^3 \overline{0}3\overline{4}4\overline{2}5 \to^2 \overline{0}3\overline{2}425 \to^1 \overline{0}31425 \to^0 031425$$

3. The array x initially contains the permutation. To express the invariant we introduce the notation

$$x_i = \text{the initial value of } x[i]$$
$$P(a,b) = \text{if } x[a] < 0 \text{ then } x[a] \equiv \overline{b} \text{ else } P(x[a], b)$$

we maintain the invariants

$$x[i] \ge 0 \quad \text{implies} \quad x_{x[i]} = i, \qquad \text{for } 0 \le i < n$$
$$P(i, x_i), \qquad \text{for } 0 \le i \le m$$

The size of x is denoted by n. The variable m starts from $n-1$ and goes down to 0. In each main iteration a negative value in the array is made positive. Therefore at the end all values will be positive and the first invariant says that we'll have the inverse permutation in x. Initially the invariants are established by assigning $x[i] = -x[i] - 1$ for all i. (The first invariant is trivially satisfied because false implies anything and the second invariant is satisfied because the 'if' branch of P is taken for all i.)

4. At each step we use one negative value $x[j]$ to figure out where to place the nonnegative m. When we write m a negative value would be overwritten if we wouldn't save it in the place j. (The value $x[j]$ is not going to be needed again anyway — it served its purpose.)

5. If we know that $P(m, x_m)$ then we can find x_m by inspecting the array x starting with $j = m$ and continuing to set $j = x[j]$ until $x[j] < 0$. (See the definition of P.) Then we can set $x[x_m] = m$, making some progress in constructing the inverse permutation. However, this might destroy the invariant $P(x_m, x_{x_m})$ (and it is possible that $x_m < m$). Since m is nonnegative $P(x_m, x_{x_m})$ is $P(m, x_{x_m})$. But $x[m]$ was just used to read x_m so it is now available to hold $\overline{x_{x_m}}$, which re-establishes the invariant.

6. The above explanation really makes sense only after you've done a few examples by hand. The proof really needs to be improved.

7. The program itself is pretty simple.

```
#include <stdio.h>
#include <stdlib.h>
  int main()
  {
    int n;    /* size of the permutation */
    int m;    /* the m appearing in the proof */
    int i, j;    /* indices */
    int *x;
    ⟨ Read the permutation 8 ⟩;
    ⟨ Compute the inverse 9 ⟩;
    ⟨ Print the inverse 10 ⟩;
  }
```

8. We trust the user with correct input. Otherwise the universe might collapse.

⟨ Read the permutation 8 ⟩ ≡
```
scanf("%d", &n);
if (n < 1) return 1;
x = (int *) malloc(n * sizeof(int));
for (i = 0; i < n; ++i) scanf("%d", &x[i]);
```
This code is used in section 7.

9. ⟨ Compute the inverse 9 ⟩ ≡
```
for (i = 0; i < n; ++i) x[i] = -x[i] - 1;
for (m = n - 1; m ≥ 0; --m) {
  for (i = m, j = x[m]; j ≥ 0; i = j, j = x[j]) ;
  x[i] = x[-j - 1], x[-j - 1] = m;
}
```
This code is used in section 7.

10. ⟨ Print the inverse 10 ⟩ ≡
```
for (i = 0; i < n; ++i) printf("%d\n", x[i]);
```
This code is used in section 7.

INVPERM2