1* Intro. Given an input file that contains a partial specification of a Boolean function of $N$ variables, this program generates clauses that are satisfiable if and only if the function has a disjunctive normal form with at most $K$ terms. Parameters $N$ and $K$ are given on the command line.

The main variables are $i+j$ (meaning that term $i$ contains $x_j$) and $i-j$ (meaning that term $i$ contains $\bar{x}_j$), for $1 \leq i \leq K$ and $1 \leq j \leq N$. There are also subsidiary variables $i,k$ for $1 \leq i \leq K$ and $1 \leq k \leq T$, if $T$ of the specified function values are true.

For example, the input file

```
101:1
001:0
100:1
111:0
011:1
```

informs us that $f(1,0,1) = 1$, $f(0,0,1) = 0$, ..., $f(0,1,1) = 1$; here $N = 3$ and $T = 3$. If we specify $K = 2$, the satisfiability problem will be satisfied, for example, by 1+1, 1–2, 2–1, 2+2; that is, $f(x_1,x_2,x_3) = x_1 \bar{x}_2 \lor \bar{x}_1 x_2$ agrees with the given specifications. [This example is taken from a paper by Kamath, Karmarker, Ramakrishnan, and Resende, Mathematical Programming 57 (1992), 215–238, where the problem is introduced and many examples are given.]

The first line of input in the example above generates seven clauses:

```
1.1 2.1 (term 1 or term 2 must be true at 101)
~1.1 ~1–1 (if term 1 is true at 101, it doesn’t contain $\bar{x}_1$)
~1.1 ~1+2 (if term 1 is true at 101, it doesn’t contain $x_2$)
~1.1 ~1–3 (if term 1 is true at 101, it doesn’t contain $\bar{x}_3$)
~2.1 ~1–1 (if term 2 is true at 101, it doesn’t contain $\bar{x}_1$)
~2.1 ~1+2 (if term 2 is true at 101, it doesn’t contain $x_2$)
~2.1 ~1–3 (if term 2 is true at 101, it doesn’t contain $\bar{x}_3$)
```

And the second line generates two:

```
1+1 1+2 1–3 (term 1 is false at 001, so it contains $x_1$, $x_2$, or $\bar{x}_3$)
2+2 2+2 2–3 (term 2 is false at 001, so it contains $x_1$, $x_2$, or $\bar{x}_3$)
```

In general, a ‘true’ line in the input generates one clause of size $K$ and $NK$ clauses of size 2; a ‘false’ line generates $K$ clauses of size $N$.

```c
#define maxn 100  /* we assume that N doesn’t exceed this */
#define O "%"  /* used for percent signs in format strings */
#include <stdio.h>
#include <stdlib.h>

char buf[maxn + 4];
int K, N, cutoff;  /* command-line parameters */
int perm_swap[] = {0, 1, 2, 0, 2, 1, 0, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 0};
int perm[] = {1,2,3,4};
int dat[4][21];

main(int argc, char *argv[])
{
    register int i, j, k, t, count;
    (Process the command line 2*);  
    printf("_sat-synth-trunc-kluj,%d,%d,%d
", N, K, cutoff);  
    (Print 24 solution-excluding lines 6*);
    t = 0;  /* this many ‘true’ lines so far */
    for (count = 0; count < cutoff ; count++) {
        if (~fgets(buf,N+4,stdin)) break;  
        (Generate clauses based on baf 3);
    }
}
```
The buffer should now hold 3.

This code is used in section 3.

```
\begin{verbatim}
(Generate clauses based on buf 3) \equiv 
if (buf[N] \neq '1' \lor buf[N + 1] < '0' \lor buf[N + 1] > '1' \lor buf[N + 2] \neq '\n' \lor buf[N + 3])
printf(stderr, "bad_input! line_1 \"O\"s'1 is ignored! \n", buf);
else 
  for (k = 0; k < N; k++)
    if (buf[k] < '0' \lor buf[k] > '1') break;
if (k < N) printf(stderr, "nonbinary data!\"O\"s'1 is ignored! \n", buf);
else if (buf[N + 1] \equiv '0') (Generate clauses for a 'false' line 4)
else (Generate clauses for a 'true' line 5);
\end{verbatim}
```

This code is used in section 1*.

```
\begin{verbatim}
(Generate clauses for a 'false' line 4) \equiv 
for (i = 1; i \leq K; i++)
  for (j = 1; j \leq N; j++) printf("O\"d"O"c"O"d", i, buf[j - 1] \equiv '0', '+' : '-', j);
printf("\n");
\end{verbatim}
```

This code is used in section 3.

```
\begin{verbatim}
(Generate clauses for a 'true' line 5) \equiv 
for (i = 1; i \leq K; i++) printf("O\"d"O"d", i, t);
printf("\n");
for (i = 1; i \leq K; i++)
  for (j = 1; j \leq N; j++)
    printf("O\"d"O\"d"O"c"O\"d\"O\"d\"O\"d\n", i, t, i, buf[j - 1] \equiv '0', '+' : '-', j);
\end{verbatim}
```

This code is used in section 3.
6* (Print 24 solution-excluding lines 6*)

```c
for (i = 0; ; i++) {
    for (j = 0; j < 4; j++)
        if (dat[j][k] > 0) printf("O^+d+"O*d_"O^+d-"O*d_"
```

This code is used in section 1*.
7* Index.
The following sections were changed by the change file: 1, 2, 6, 7.

argc: 1* 2*
argv: 1* 2*
buf: 1* 3, 4, 5.
count: 1*
cutoff: 1* 2*
dat: 1* 6*
ext: 2*
fgets: 1*
fprintf: 2* 3.
i: 1*
j: 1*
K: 1*
k: 1*
main: 1*
maxn: 1* 2*
N: 1*
O: 1*
perm: 1* 6*
perm_swap: 1* 6*
printf: 1* 4, 5, 6*
sscanf: 2*
stderr: 2* 3.
stdin: 1*
t: 1*
(Generate clauses based on \textit{buf} 3) Used in section 1*.
(Generate clauses for a ‘false’ line 4) Used in section 3.
(Generate clauses for a ‘true’ line 5) Used in section 3.
(Print 24 solution-excluding lines 6*) Used in section 1*.
(Process the command line 2*) Used in section 1*.