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 $ar{x}_j$), for $1 \leq i \leq K$ and $1 \leq j \leq N$. There also are 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)
- $\neg 1.1 \ \neg 1-1$ (if term 1 is true at 101, it doesn’t contain $x_1$)
- $\neg 1.1 \ \neg 1+2$ (if term 1 is true at 101, it doesn’t contain $x_2$)
- $\neg 1.1 \ \neg 1-3$ (if term 1 is true at 101, it doesn’t contain $x_3$)
- $\neg 2.1 \ \neg 1-1$ (if term 2 is true at 101, it doesn’t contain $x_1$)
- $\neg 2.1 \ \neg 1+2$ (if term 2 is true at 101, it doesn’t contain $x_2$)
- $\neg 2.1 \ \neg 1-3$ (if term 2 is true at 101, it doesn’t contain $x_3$)

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 $x_3$)
- $2+1 \ 2+2 \ 2-3$ (term 2 is false at 001, so it contains $x_1$, $x_2$, or $x_3$)

In general, a ‘true’ line in the input generates one clause of size $K$ and $N K$ 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>

int buf[maxn + 4];

main(int argc, char *argv[]) {
    register int i, j, k, t;
    (Process the command line 2);
    printf("sat-synth,%d,%d\n", N, K);
    t = 0;  /* this many 'true' lines so far */
    while (1) {
        if (!fgets(buf, N + 4, stdin)) break;
        (Generate clauses based on buf 3);
    }
}
```
2. \( \text{Process the command line 2) } \equiv \)

if \((\text{argc} \neq 3 \lor \text{sscanf(\text{argv}[1], "O"d", \&N) \neq 1 \lor \text{sscanf(\text{argv}[2], "O"d", \&K) \neq 1}) \) {
    fprintf(stderr, "Usage:\"O"s\&N\&K\n", \text{argv}[0]);
    exit(-1);
}

if \((N > \text{maxn}) \) {
    fprintf(stderr, "That \"O"d\& is too big for me, I'm set up for at most \"O"d!\n", N, \text{maxn});
    exit(-2);
}

This code is used in section 1.

3. The buffer should now hold \(N\) digits, then colon, digit, '\n', and '0'.

(Generate clauses based on buf 3) \( \equiv \)

if \((\text{buf}[N] \neq \:'\:' \lor \text{buf}[N + 1] < \:'0' \lor \text{buf}[N + 1] > \:'1' \lor \text{buf}[N + 2] \neq \:\'n' \lor \text{buf}[N + 3]) \) {
    fprintf(stderr, "\text{bad input line \"O"s\& is ignored!\n", buf}));
} else if \((\text{buf}[N + 1] \equiv \:'0') \) (Generate clauses for a 'false' line 4)

else (Generate clauses for a 'true' line 5);

This code is used in section 1.

4. (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, \text{buf}[j - 1] \equiv \:'0' \lor \:'+' \lor \:'-', j); printf("\n");
}

This code is used in section 3.

5. (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 \((j = 1; j \leq N; j++) \) printf("\"O"d\&"O"c\"O"d\&\n", i, t, i, \text{buf}[j - 1] \equiv \:'0' \lor \:'+' \lor \:'-', j));

This code is used in section 3.
6. Index.

argc: 1, 2.
argv: 1, 2.
buf: 1, 3, 4, 5.
exit: 2.
fgets: 1.
fprintf: 2, 3.
i: 1.
j: 1.
K: 1.
k: 1.
main: 1.
maxn: 1, 2.
N: 1.
O: 1.
printf: 1, 4, 5.
sscanf: 2.
stderr: 2, 3.
stdin: 1.
t: 1.
(Generate clauses based on $buf$ 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.
(Process the command line 2) Used in section 1.
SAT-SYNTH

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intro</td>
<td>1</td>
</tr>
<tr>
<td>Index</td>
<td>6</td>
</tr>
</tbody>
</table>