1. **Intro.** This little program outputs clauses that are satisfiable if and only if the graph $g$ can be $c$-colored with kernels, given $g$ and $c$.

   (It generalizes SAT-PIGEONS, which is the case where $g = K_m$ and $c = n$.)

   Suppose the graph has $m$ edges and $n$ vertices. Then there are $nc$ variables $v.k$, meaning that vertex $v$ gets color $k$. And there are $n$ clauses of size $c$ (to ensure that each vertex gets at least one color), plus $mc$ clauses of size $2$ (to ensure that adjacent vertices don’t share a color). Plus $nc$ clauses for each extended neighborhood.

```c
#include <stdio.h>
#include <stdlib.h>
#include "gb_graph.h"
#include "gb_save.h"

int c;

main(int argc, char *argv[]) {
    register int i, j, k;
    register Arc *a;
    register Graph *g;
    register Vertex *v;
    (Process the command line 2);
    (Generate the positive clauses 3);
    (Generate the negative clauses 4);
    (Generate the kernel clauses 5);
}

2. (Process the command line 2) $\equiv$
   if (argc $\neq$ 3 $\lor$ sscanf(argv[2], "%d", &c) $\neq$ 1) {
       fprintf(stderr, "Usage: \%sat-color-kernel \%s \%d
", argv[0]);
       exit(-1);
   }

   g = restore_graph(argv[1]);
   if (!g) {
       fprintf(stderr, "I couldn't reconstruct graph \%s!\n", argv[1]);
       exit(-2);
   }
   if (c $\leq$ 0) {
       fprintf(stderr, "c must be positive!\n");
       exit(-3);
   }

   printf("\%sat-color-kernel\%s\%d\n", argv[1], c);

   This code is used in section 1.

3. (Generate the positive clauses 3) $\equiv$
   for ($v = g$-vertices; $v < g$-vertices $+ g$-n; $v++$) {
      for ($k = 1; k \leq c; k++$) printf("\%s \%d", v-name, k);
      printf("\n");
   }

   This code is used in section 1.
4. (Generate the negative clauses 4) \( k = 1; k \leq c; k++ \)
   \textbf{for} (\( v = g\)-vertices; \( v < g\)-vertices + \( g\)-n; \( v++ \))
   \textbf{for} (\( a = v\)-\( a\)res; \( a = a\)-\( a\)next)
   \textbf{if} (\( a\)-\( a\)tip > \( v \)) \textbf{printf} ("~%s.%d ~%s.%d\n"; \( v\)-\( v\)name, \( k \); \( a\)-\( a\)tip\-\( a\)name, \( k \));
This code is used in section 1.

5. (Generate the kernel clauses 5) \( k = 1; k \leq c; k++ \)
   \textbf{for} (\( v = g\)-vertices; \( v < g\)-vertices + \( g\)-n; \( v++ \)) \{ 
   \textbf{printf} ("%s.%d", \( v\)-\( v\)name, \( k \));
   \textbf{for} (\( a = v\)-\( a\)res; \( a = a\)-\( a\)next) \textbf{printf} ("\%s.%d", \( a\)-\( a\)tip\-\( a\)name, \( k \));
   \textbf{printf} ("\n");
\}
This code is used in section 1.
6. Index.

a: 1.
Arc: 1.
arc: 4, 5.
argc: 1, 2.
argv: 1, 2.
c: 1.
exit: 2.
fprintf: 2.
g: 1.
Graph: 1.
i: 1.
j: 1.
k: 1
main: 1
name: 3, 4, 5.
next: 4, 5.
printf: 2, 3, 4, 5.
restore_graph: 2.
sscanf: 2.
stderr: 2.
tip: 4, 5.
v: 1.
Vertex: 1.
vertices: 3, 4, 5.
(Generate the kernel clauses 5) Used in section 1.
(Generate the negative clauses 4) Used in section 1.
(Generate the positive clauses 3) Used in section 1.
(Process the command line 2) Used in section 1.