**1. Intro.** This simple program creates the line graph of the "flower snark" $J_n$ of order $n$, given $n$ on the command line. The vertices of $J_n$ are $t_j$, $u_j$, $v_j$, $w_j$ for $1 \le j \le n$; the edges are $t_j \longrightarrow t_{j+1}$, $t_j \longrightarrow u_j$, $u_j \longrightarrow v_j$, $u_j \longrightarrow w_j$, $v_j \longrightarrow w_{j+1}$, $w_j \longrightarrow v_{j+1}$, with subscripts treated modulo $n$. The vertices of $L(J_n)$ are conveniently named $a_j$, $b_j$, $c_j$, $d_j$, $e_j$, $f_j$, in correspondence with those edges.

```
#include "gb_graph.h"      /* we use the GB_GRAPH data structures */
#include "gb_save.h"       /* and we save our results in ASCII format */
  int n;      /* the order */
  char buf[16];

  int main(int argc, char *argv[])
  {
    register int i, j, k;
    register Graph *g;

    ⟨Read the command line to determine n 2⟩;
    ⟨Create an empty graph of 6n vertices, and name them 3⟩;
    for (j = 1; j ≤ n; j++) ⟨Generate edges that depend on j 4⟩;
    sprintf(g→id, "flowersnarkline(%d)", n);
    sprintf(buf, "fsnarkline%d.gb", n);
    save_graph(g, buf);
  }
```

**2.** ⟨Read the command line to determine $n$ 2⟩ ≡

```
  if (argc ≠ 2 ∨ sscanf(argv[1], "%d", &n) ≠ 1) {
    fprintf(stderr, "Usage:␣%s␣n\n", argv[0]);
    exit(−1);
  }
```

This code is used in section 1.

**3.**

```
#define avert(j) (g→vertices + (6 ∗ (j) − 6))
#define bvert(j) (g→vertices + (6 ∗ (j) − 5))
#define cvert(j) (g→vertices + (6 ∗ (j) − 4))
#define dvert(j) (g→vertices + (6 ∗ (j) − 3))
#define evert(j) (g→vertices + (6 ∗ (j) − 2))
#define fvert(j) (g→vertices + (6 ∗ (j) − 1))
```

⟨ Create an empty graph of $6n$ vertices, and name them 3 ⟩ ≡

```
  g = gb_new_graph(6 ∗ n);
  if (¬g) {
    fprintf(stderr, "Can't␣create␣an␣empty␣graph␣of␣%d␣vertices!\n", 6 ∗ n);
    exit(−2);
  }
  for (j = 1; j ≤ n; j++) {
    sprintf(buf, "a%d", j);
    avert(j)→name = gb_save_string(buf);
    sprintf(buf, "b%d", j);
    bvert(j)→name = gb_save_string(buf);
    sprintf(buf, "c%d", j);
    cvert(j)→name = gb_save_string(buf);
    sprintf(buf, "d%d", j);
    dvert(j)→name = gb_save_string(buf);
    sprintf(buf, "e%d", j);
    evert(j)→name = gb_save_string(buf);
    sprintf(buf, "f%d", j);
    fvert(j)→name = gb_save_string(buf);
  }
```

This code is used in section 1.

**4.**

```
#define incr(j) ((j) ≡ n ? 1 : (j) + 1)
```

⟨ Generate edges that depend on $j$ 4 ⟩ ≡

```
  {
    gb_new_edge(avert(j), avert(incr(j)), 1);
    gb_new_edge(avert(j), bvert(j), 1);
    gb_new_edge(avert(j), bvert(incr(j)), 1);
    gb_new_edge(bvert(j), cvert(j), 1);
    gb_new_edge(bvert(j), dvert(j), 1);
    gb_new_edge(cvert(j), dvert(j), 1);
    gb_new_edge(cvert(j), evert(j), 1);
    gb_new_edge(dvert(j), fvert(j), 1);
    gb_new_edge(evert(j), dvert(incr(j)), 1);
    gb_new_edge(evert(j), fvert(incr(j)), 1);
    gb_new_edge(fvert(j), cvert(incr(j)), 1);
    gb_new_edge(fvert(j), evert(incr(j)), 1);
  }
```

This code is used in section 1.

## 5. Index.

⟨ Create an empty graph of $6n$ vertices, and name them 3 ⟩ Used in section 1.
⟨ Generate edges that depend on $j$ 4 ⟩ Used in section 1.
⟨ Read the command line to determine $n$ 2 ⟩ Used in section 1.

# FLOWER-SNARK-LINE