1. Random graphs. This is a short program for generating inputs for Sedgewick s programs in *Algorithms in C* (Parts 1–4), Section 1.3. The user should use the command-line options `-nN -nM -sS` to ask for a random graph with `N` vertices and `M` edges. The quantity `S` is used as the seed for the random number generator.

```
#include "gb_graph.h"    /* we use the GB_GRAPH data structures */
#include "gb_rand.h"     /* for the random graph generation routine */
#include "gb_sort.h"     /* for SGB's sorting routine */
#include "gb_flip.h"     /* for SGB's random number generator */
  ⟨ Declarations 3 ⟩
  ⟨ The program 2 ⟩
```

2. ⟨ The program 2 ⟩ ≡

```
  int main(int argc, char *argv[])
  { Graph *g;
    unsigned long n = 10;
    unsigned long m = 20;
    long seed = 0;
    ⟨ Scan the command-line options 7 ⟩;
    g = random_graph(n, m, 0, 0, 0, Λ, Λ, 1, 1, seed);
    ⟨ Create the linked list representing the edges of g 4 ⟩;
    ⟨ Sort and output the pairs of endpoints of edges 5 ⟩;
    return 0;    /* normal exit */
  }
```

This code is used in section 1.

3. ⟨ Declarations 3 ⟩ ≡

```
  typedef struct node_struct {
    long key;    /* a numeric quantity, assumed nonnegative */
    struct node_struct *link;    /* the next node on a list */
    long u, w;    /* the vertex names */
  } node;
  node *ll_edges = Λ;    /* pointer to the beginning of the linked list of edge clones */
```

This code is used in section 1.

4. ⟨ Create the linked list representing the edges of g 4 ⟩ ≡

```
if (g ≡ Λ) printf("Something␣went␣wrong␣(panic␣code␣%ld)!\n", panic_code);
else {
  register Vertex *v;   /* current vertex being visited */
  for (v = g→vertices; v < g→vertices + g→n; v++) {
    register Arc *a;   /* current arc from v */
    for (a = v→arcs; a; a = a→next)
      if (a→tip > v) {
        register node *p = malloc(sizeof *p);
        p→u = atoi(v→name);
        p→w = atoi(a→tip→name);
        p→key = gb_next_rand();
        p→link = ll_edges;
        ll_edges = p;
      }
  }
}
```

This code is used in section 2.

5. ⟨ Sort and output the pairs of endpoints of edges 5 ⟩ ≡

```
gb_linksort(ll_edges);
⟨ Output the pairs 6 ⟩
```

This code is used in section 2.

6. ⟨ Output the pairs 6 ⟩ ≡

```
{ int j;
  node *p;
  for (j = 127; j ≥ 0; j--)
    for (p = (node *) gb_sorted[j]; p; p = p→link) printf("␣%ld␣%ld\n", p→u, p→w);
}
```

This code is used in section 5.

7. ⟨ Scan the command-line options 7 ⟩ ≡

```
while (--argc) {
  if (sscanf(argv[argc], "-n%lu", &n) ≡ 1) ;
  else if (sscanf(argv[argc], "-m%lu", &m) ≡ 1) ;
  else if (sscanf(argv[argc], "-s%ld", &seed) ≡ 1) ;
  else {
    fprintf(stderr, "Usage:␣%s␣[-nN][-mM][-sS]\n", argv[0]);
    return -2;
  }
}
```

This code is used in section 2.

8. Index.

RANDOM2