1*. Introduction. This is a hastily written implementation of the daghull algorithm.

```
  format Graph int      /* gb_graph defines the Graph type and a few others */
  format Vertex int
  format Arc int
  format Area int
#include "gb_graph.h"
#include "gb_rand.h"
#include "gb_miles.h"
  int n = 128;
  ⟨Global variables 2⟩
  ⟨Procedures 13*⟩
  main(argc, argv)
      int argc;
      char **argv;
  {
    ⟨Local variables 7⟩
    Graph *g;
    int kk;
    char str[10];
    if (argc ≠ 2) n = 100;
    else if (sscanf(argv[1], "%d", &n) ≠ 1) {
      printf("Usage:␣%s␣[n]\n", argv[0]); exit(1);
    }
    else if (n < 20) {
      printf("n␣should␣be␣at␣least␣20!\n"); exit(1);
    }
    g = gb_new_graph(n);
    gb_init_rand(0);
    for (kk = 0, v = g→vertices; kk < n; kk++, v++) {
      sprintf(str, "%d", kk);
      v→name = gb_save_string(str);
      v→x.I = gb_next_rand() & #ffff;
      v→y.I = gb_next_rand() & #ffff;
      if (n < 150) printf("point␣%s=(%d,%d)\n", v→name, v→x.I, v→y.I);
    }
    mems = ccs = 0;
    ⟨Find convex hull of g 8⟩;
    printf("Total␣of␣%d␣mems␣and␣%d␣calls␣on␣ccw.\n", mems, ccs);
  }
```

13*. Determinants. I need code for the primitive function *ccw*. Floating-point arithmetic suffices for my purposes.

We want to evaluate the determinant

$$ccw(u,v,w)=\begin{vmatrix} u(x) & u(y) & 1\\ v(x) & v(y) & 1\\ w(x) & w(y) & 1\end{vmatrix}=\begin{vmatrix} u(x)-w(x) & u(y)-w(y)\\ v(x)-w(x) & v(y)-w(y)\end{vmatrix}.$$

```
⟨ Procedures 13* ⟩ ≡
  int ccw(u, v, w)
      Vertex *u, *v, *w;
  { register double wx = (double) w→x.I, wy = (double) w→y.I;
    register double det = ((double) u→x.I − wx) * ((double) v→y.I − wy) − ((double) u→y.I − wy) *
        ((double) v→x.I − wx);
    Vertex *uu = u, *vv = v, *ww = w, *t;
    if (det ≡ 0) {
      det = 1;
      if (u→x.I > v→x.I ∨ (u→x.I ≡ v→x.I ∧ (u→y.I > v→y.I ∨ (u→y.I ≡ v→y.I ∧ u→z.I > v→z.I)))) {
        t = u; u = v; v = t; det = −det;
      }
      if (v→x.I > w→x.I ∨ (v→x.I ≡ w→x.I ∧ (v→y.I > w→y.I ∨ (v→y.I ≡ w→y.I ∧ v→z.I > w→z.I)))) {
        t = v; v = w; w = t; det = −det;
      }
      if (u→x.I > v→x.I ∨ (u→x.I ≡ v→x.I ∧ (u→y.I > v→y.I ∨ (u→y.I ≡ v→y.I ∧ u→z.I < v→z.I)))) {
        det = −det;
      }
    }
    if (n < 150)
      printf("cc(%s;␣%s;␣%s)␣is␣%s\n", uu→name, vv→name, ww→name, det > 0 ? "true" : "false");
    ccs++;
    return (det > 0);
  }
```

This code is used in section 1*.

The following sections were changed by the change file: 1, 13.

⟨ Compile two new instructions, for (u, vv) and (vv, v) 12 ⟩ Used in section 11.
⟨ Find convex hull of g 8 ⟩ Used in section 1*.
⟨ Follow the instructions; **continue** if vv is inside the current hull 10 ⟩ Used in section 8.
⟨ Global variables 2, 5 ⟩ Used in section 1*.
⟨ Initialize the array of instructions 4 ⟩ Used in section 6.
⟨ Initialize the data structures 6 ⟩ Used in section 8.
⟨ Local variables 7 ⟩ Used in section 1*.
⟨ Print the convex hull 9 ⟩ Used in section 8.
⟨ Procedures 13* ⟩ Used in section 1*.
⟨ Update the convex hull, knowing that vv lies outside the consecutive hull vertices u and v 11 ⟩ Used in section 8.