1. Intro. We output an Eulerian trail of the (undirected) graph named on the command line. (Each edge is considered to be two directed arcs; thus it is traversed in both directions.)

If the graph isn't connected, we consider only the vertices that are reachable from the first one, *g→vertices*.

```
#include <stdio.h>
#include <stdlib.h>
#include "gb_graph.h"
#include "gb_save.h"
  ⟨ Subroutine 3 ⟩
  main(int argc, char *argv[])
  {
    register int k;
    Graph *g;
    Vertex *u, *v;
    Arc *a;
    ⟨ Input the graph 2 ⟩;
    ⟨ Traverse depth first 4 ⟩;
    ⟨ Output the trail 5 ⟩;
    printf("\n");
  }
```

2. ⟨ Input the graph 2 ⟩ ≡

```
  if (argc ≠ 2 ∨ ¬(g = restore_graph(argv[1]))) {
    fprintf(stderr, "Usage:␣%s␣foo.gb\n", argv[0]);
    exit(−1);
  }
  fprintf(stderr, "OK,␣I've␣input␣`%s'.\n", argv[1]);
  gb_new_edge(g→vertices, g→vertices + g→n, 0);      /* dummy edge */
```

This code is used in section 1.

3. Subroutine *dfs*(*u*, *v*) sets *v→parent* = *u* and *v→nav* to the vertex that follows *u* in *v*'s adjacency list. It also explores all vertices reachable from *v* that haven't already been seen.

```
#define parent v.V
#define nav w.A
⟨ Subroutine 3 ⟩ ≡
  void dfs(register Vertex *u, register Vertex *v)
  {
    register Vertex *w;
    register Arc *a;
    v→parent = u;
    for (a = v→arcs; a; a = a→next) {
      w = a→tip;
      if (w ≡ u) v→nav = a→next;
      else if (w→parent ≡ Λ) dfs(v, w);
    }
  }
```

This code is used in section 1.

4. ⟨ Traverse depth first 4 ⟩ ≡

```
  dfs(g→vertices + g→n, g→vertices);
```

This code is used in section 1.

5. Now the Eulerian traversal is beautifully simple.

⟨ Output the trail 5 ⟩ ≡

```
for (v = g→vertices; v ≠ g→vertices + g→n; ) {
  printf("␣%s", v→name);
  a = v→nav;
  if (¬a) a = v→arcs;
  v→nav = a→next;
  v = a→tip;
}
```

This code is used in section 1.

6. Index.

EULER-TRAIL