Important: Before reading G_ROGET_SIMPLES, please read or at least skim the program for GB_ROGET.

1. Some graphs for strong component computation. I've cut out parts of a couple of programs in the Stanford GraphBase to put together this source. You can generate some interesting directed graphs with this program to check your programs for finding strong components. You should read *gb_roget* to understand how these graphs are generated. The accompanying program *g_roget_simples* generates the same graphs as this program, but prints out the vertices with numerical identifiers. Try both programs to understand what I mean.

2. We permit command-line options in typical UNIX style so that a variety of graphs can be studied: The user can say '`-n`⟨number⟩', '`-d`⟨number⟩', '`-p`⟨number⟩', and/or '`-s`⟨number⟩' to change the default values of the parameters in the graph $roget(n, d, p, s)$.

The special command-line option `-g`⟨filename⟩ overrides all others. It substitutes an external graph previously saved by *save_graph*.

```
#include "gb_graph.h"      /* the GraphBase data structures */
#include "gb_roget.h"      /* the roget routine */
#include "gb_save.h"       /* restore_graph */
  ⟨ Preprocessor definitions ⟩
  main(argc, argv)
      int argc;      /* the number of command-line arguments */
      char *argv[];      /* an array of strings containing those arguments */
  { Graph *g;      /* the graph we will work on */
    register Vertex *v;      /* the current vertex of interest */
    unsigned long n = 0;      /* the desired number of vertices (0 means infinity) */
    unsigned long d = 0;      /* the minimum distance between categories in arcs */
    unsigned long p = 0;      /* 65536 times the probability of rejecting an arc */
    long s = 0;      /* the random number seed */
    char *filename = Λ;      /* external graph substituted for roget */
    ⟨ Scan the command-line options 3 ⟩;
    g = (filename ? restore_graph(filename) : roget(n, d, p, s));
    if (g ≡ Λ) {
      fprintf(stderr, "Sorry, can't create the graph! (error code %ld)\n", panic_code);
      return −1;
    }
    ⟨ Print the vertices and edges of g 5 ⟩;
    return 0;      /* normal exit */
  }
```

3. ⟨ Scan the command-line options 3 ⟩ ≡

```
  while (−−argc) {
    if (sscanf(argv[argc], "-n%lu", &n) ≡ 1) ;
    else if (sscanf(argv[argc], "-d%lu", &d) ≡ 1) ;
    else if (sscanf(argv[argc], "-p%lu", &p) ≡ 1) ;
    else if (sscanf(argv[argc], "-s%ld", &s) ≡ 1) ;
    else if (strncmp(argv[argc], "-g", 2) ≡ 0) filename = argv[argc] + 2;
    else {
      fprintf(stderr, "Usage: %s [-nN] [-dN] [-pN] [-sN] [-gfoo]\n", argv[0]);
      return −2;
    }
  }
```

This code is used in section 2.

4. Printing out the graph. We print out the graph in a rather simple way: we just print the adjacency lists. For simplicity, the vertices are identified with the integers $0, \ldots, n-1$, where n is the number of vertices in the graph g.

5. ⟨ Print the vertices and edges of g 5 ⟩ ≡

```
if (g ≡ Λ) printf("Something␣went␣wrong␣(panic␣code␣%ld)!\n", panic_code);
else {
  register Vertex *v;      /* current vertex being visited */
  printf("The␣graph␣whose␣official␣name␣is\n\n␣␣%s\n\n", g→id);
  printf("has␣%ld␣vertices␣and␣%ld␣arcs:\n\n", g→n, g→m);
  for (v = g→vertices; v < g→vertices + g→n; v++) {
    register Arc *a;      /* current arc from v */
    printf("\n%ld:", v − g→vertices);
    for (a = v→arcs; a; a = a→next) printf("␣%ld", a→tip − g→vertices);
  }
  printf("\n");
}
```

This code is used in section 2.

6. Index. We close with a list that shows where the identifiers of this program are defined and used.

⟨ Print the vertices and edges of g 5 ⟩ Used in section 2.
⟨ Scan the command-line options 3 ⟩ Used in section 2.

April 25, 2007 at 21:34

G_ROGET_SIMPLES