

1. Intro. A trivial program to create an SGB graph. The first line of standard input lists the vertex names; the remaining lines list the (directed) edges, as triples $x\ y\ d$.

An optional command-line argument gives the name of the graph. For example, if the name is `test`, the graph is saved as `/tmp/test.gb`.

```
#define maxn 100000 /* at most this many vertices */
#define maxl 3 /* maximum length of vertex name */
#define bufsize (maxl + 1) * maxn + 2

#include <stdio.h>
#include <stdlib.h>
#include "gb_graph.h"
#include "gb_save.h"
char buf[bufsize + 1];
char names[maxn][maxl + 1];
char nbuf[maxl + 1];
char filenamebuf[ID_FIELD_SIZE + 8] = "/tmp/makegraph.gb";
int main(int argc, char *argv[])
{
    register int j, k, m, n, d;
    Graph *g;
    Vertex *u, *v;

    <Input the vertices 2>;
    <Input the edges 3>;
    <Output the graph 4>;
}

2. <Input the vertices 2> ≡
buf[bufsize] = '\n';
if (!fgets(buf, bufsize, stdin)) {
    fprintf(stderr, "Couldn't read the variable-name line!\n");
    exit(-1);
}
for (n = k = 0; n < maxn; n++) {
    while (buf[k] == ' ') k++;
    if (buf[k] == '\n') break;
    for (j = 0; buf[k] != ' ' & buf[k] != '\n' & j ≤ maxl; j++, k++) names[n][j] = buf[k];
    if (j > maxl) {
        fprintf(stderr, "Vertex name is too long! %s", buf - k - j);
        exit(-2);
    }
}
g = gb_new_graph(n);
for (k = 0; k < n; k++) (g->vertices + k)-name = gb_save_string(names[k]);
hash_setup(g);
printf("I've created a graph with %d vertices...\n", n);
```

This code is used in section 1.

3. $\langle \text{Input the edges } 3 \rangle \equiv$

```

for ( $m = 0$ ; ;  $m++$ ) {
    if ( $\neg fgets(buf, bufsize, stdin)$ ) break;
    for ( $k = 0$ ;  $buf[k] \equiv '\_'$ ;  $k++$ ) ;
    for ( $j = 0$ ;  $buf[k] \neq '\_'$   $\wedge j < maxl$ ;  $j++, k++$ )  $nbuf[j] = buf[k]$ ;
     $nbuf[j] = '\0'$ ;
     $u = hash\_out(nbuf)$ ;
    if ( $\neg u$ ) {
         $fprintf(stderr, "Unknown\_first\_vertex:\_s", buf)$ ;
         $exit(-3)$ ;
    }
    for ( ;  $buf[k] \equiv '\_'$ ;  $k++$ ) ;
    for ( $j = 0$ ;  $buf[k] \neq '\_'$   $\wedge j < maxl$ ;  $j++, k++$ )  $nbuf[j] = buf[k]$ ;
     $nbuf[j] = '\0'$ ;
     $v = hash\_out(nbuf)$ ;
    if ( $\neg v$ ) {
         $fprintf(stderr, "Unknown\_second\_vertex:\_s", buf)$ ;
         $exit(-4)$ ;
    }
    for ( ;  $buf[k] \equiv '\_'$ ;  $k++$ ) ;
    for ( $d = 0$ ;  $buf[k] \geq '0' \wedge buf[k] \leq '9'$ ;  $k++$ )  $d = 10 * d + buf[k] - '0'$ ;
     $gb\_new\_arc(u, v, d)$ ;
}
 $printf("\_and\_d\_arcs...\_n", m)$ ;

```

This code is used in section 1.

4. $\langle \text{Output the graph } 4 \rangle \equiv$

```

if ( $argc > 1$ ) {
     $sprintf(g\_id, "\_s", ID\_FIELD\_SIZE - 1, argv[1])$ ;
     $sprintf(filenamebuf, "/tmp/\_s.gb", ID\_FIELD\_SIZE - 1, argv[1])$ ;
}
 $save\_graph(g, filenamebuf)$ ;
 $printf("\_and\_file\_s\_holds\_the\_result.\_n", filenamebuf)$ ;

```

This code is used in section 1.

5. Index.

argc: [1](#), [4](#).
argv: [1](#), [4](#).
buf: [1](#), [2](#), [3](#).
bufsize: [1](#), [2](#), [3](#).
d: [1](#).
exit: [2](#), [3](#).
fgets: [2](#), [3](#).
filenamebuf: [1](#), [4](#).
fprintf: [2](#), [3](#).
g: [1](#).
gb_new_arc: [3](#).
gb_new_graph: [2](#).
gb_save_string: [2](#).
Graph: [1](#).
hash_out: [3](#).
hash_setup: [2](#).
id: [4](#).
ID_FIELD_SIZE: [1](#), [4](#).
j: [1](#).
k: [1](#).
m: [1](#).
main: [1](#).
maxl: [1](#), [2](#), [3](#).
maxn: [1](#), [2](#).
n: [1](#).
name: [2](#).
names: [1](#), [2](#).
nbuf: [1](#), [3](#).
printf: [2](#), [3](#), [4](#).
save_graph: [4](#).
sprintf: [4](#).
stderr: [2](#), [3](#).
stdin: [2](#), [3](#).
u: [1](#).
v: [1](#).
Vertex: [1](#).
vertices: [2](#).

⟨Input the edges 3⟩ Used in section 1.
⟨Input the vertices 2⟩ Used in section 1.
⟨Output the graph 4⟩ Used in section 1.

MAKEDIGRAPH

	Section	Page
Intro	1	1
Index	5	3