

(Downloaded from <https://cs.stanford.edu/~knuth/programs.html> and typeset on September 17, 2017)

1. Introduction. This program takes an algebraic specification of a series-parallel graph and converts it to Stanford GraphBase format.

The given graph is specified using a simple right-Polish syntax

$$G \rightarrow - \mid GG\mathbf{s} \mid GG\mathbf{p}$$

so that, for example, the specifications `----ps-sp--sp` and `----p-ss--spp` both denote the graph

(The conventions are identical to those of SPSPAN, so that I can compare that program with GRAYSPAN.)

```
#include "gb_graph.h"
#include "gb_save.h"
  ⟨Preprocessor definitions⟩
  ⟨Global variables 3⟩
  ⟨Subroutines 7⟩
main(int argc, char *argv[])
{
  register int j, k;
  if (argc ≠ 3) {
    fprintf(stderr, "Usage: %s SPformula.foo.gb\n", argv[0]); exit(0);
  }
  ⟨Parse the formula argv[1] into a binary tree 2⟩;
  ⟨Convert the binary tree to a graph 6⟩;
  k = save_graph(g, argv[2]);
  if (k) printf("I had trouble saving in %s (anomalies %x)!\n", argv[2], k);
  else printf("Graph %s saved successfully in %s.\n", g-id, argv[2]);
}
```

2. In the following code, we have scanned j binary operators (including jj of type **s**) and there are k items on the stack.

```
#define abort(mess)
  { fprintf(stderr, "Parsing error: %.*s | %s, %s!\n", p - argv[1], argv[1], p, mess); exit(-1); }
  ⟨Parse the formula argv[1] into a binary tree 2⟩ ≡
  {
    register char *p = argv[1];
    for (j = k = 0; *p; p++)
      if (*p ≡ '-') ⟨Create a new leaf 4⟩
      else if (*p ≡ 's' ∨ *p ≡ 'p') ⟨Create a new branch 5⟩
      else abort("bad symbol");
    if (k ≠ 1) abort("disconnected graph");
  }
```

This code is used in section 1.

3. **#define** *maxn* 1000 /* the maximum number of leaves; *not checked* */

⟨Global variables 3⟩ ≡

```
int stack[maxn]; /* stack for parsing */
int llink[maxn], rlink[maxn]; /* binary subtrees */
char buffer[8]; /* for sprinting */
int jj;
Graph *g;
```

This code is used in section 1.

4. ⟨Create a new leaf 4⟩ ≡

```
stack[k++] = 0;
```

This code is used in section 2.

5. ⟨Create a new branch 5⟩ ≡

```
{
    if (k < 2) abort("missing_operand");
    rlink[++j] = stack[--k];
    llink[j] = stack[k - 1];
    if (*p == 's') jj++;
    stack[k - 1] = (*p == 's' ? #100 : 0) + j;
}
```

This code is used in section 2.

6. Now we convert the binary tree to the desired graph, working top down.

#define *vert*(*k*) (*g*-vertices + (*k*))

⟨Convert the binary tree to a graph 6⟩ ≡

```
g = gb_new_graph(jj + 2);
if (!g) {
    fprintf(stderr, "Can't create the graph!\n");
    exit(-1);
}
sprintf(g-id, "SP%.152s", argv[1]);
for (k = 0; k < g-n; k++) {
    sprintf(buffer, "%d", k);
    vert(k)-name = gb_save_string(buffer);
}
build(stack[0], 0, 1);
```

This code is used in section 1.

7. A recursive subroutine called *build* governs the construction process.

⟨Subroutines 7⟩ ≡

```
void build(int stackitem, int lft, int rt)
{
    register int t, j;
    if (stackitem == 0) gb_new_edge(vert(lft), vert(rt), 0);
    else {
        t = stackitem >> 8, j = stackitem & #ff; /* type and location of a binary op */
        if (t) t = --jj + 2, build(llink[j], lft, t), build(rlink[j], t, rt);
        else build(llink[j], lft, rt), build(rlink[j], lft, rt);
    }
}
```

This code is used in section 1.

8. Index.

abort: [2](#), [5](#).
argc: [1](#).
argv: [1](#), [2](#), [6](#).
buffer: [3](#), [6](#).
build: [6](#), [7](#).
exit: [1](#), [2](#), [6](#).
fprintf: [1](#), [2](#), [6](#).
g: [3](#).
gb_new_edge: [7](#).
gb_new_graph: [6](#).
gb_save_string: [6](#).
Graph: [3](#).
id: [1](#), [6](#).
j: [1](#), [7](#).
jj: [2](#), [3](#), [5](#), [6](#), [7](#).
k: [1](#).
lft: [7](#).
llink: [3](#), [5](#), [7](#).
main: [1](#).
maxn: [3](#).
mess: [2](#).
name: [6](#).
p: [2](#).
printf: [1](#).
rlink: [3](#), [5](#), [7](#).
rt: [7](#).
save_graph: [1](#).
sprintf: [6](#).
stack: [3](#), [4](#), [5](#), [6](#).
stackitem: [7](#).
stderr: [1](#), [2](#), [6](#).
t: [7](#).
vert: [6](#), [7](#).
vertices: [6](#).

- ⟨ Convert the binary tree to a graph 6 ⟩ Used in section 1.
- ⟨ Create a new branch 5 ⟩ Used in section 2.
- ⟨ Create a new leaf 4 ⟩ Used in section 2.
- ⟨ Global variables 3 ⟩ Used in section 1.
- ⟨ Parse the formula $argv[1]$ into a binary tree 2 ⟩ Used in section 1.
- ⟨ Subroutines 7 ⟩ Used in section 1.

SPGRAPH

	Section	Page
Introduction	1	1
Index	8	3