

```
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: }
```

```
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) + 1) * sizeof(char));
30:     if (s2 == NULL)
31:         return 1;
32:
33:     // copy string
34:     int n = strlen(s1);
35:     for (int i = 0; i < n; i++)
36:         s2[i] = s1[i];
37:     s2[n] = '\0';
38:
39:     // change copy
40:     printf("Capitalizing copy...\n");
41:     if (strlen(s2) > 0)
42:         s2[0] = toupper(s2[0]);
43:
44:     // print original and copy
45:     printf("Original: %s\n", s1);
46:     printf("Copy:      %s\n", s2);
47:
48:     // free memory
49:     free(s1);
50:     free(s2);
51: }
```

```

1: /*****
2:  * cs50.c
3:  *
4:  * version 1.1.3
5:  *
6:  * Computer Science 50
7:  * Glenn Holloway
8:  * David J. Malan
9:  *
10: * Definitions for CS 50's library.
11: * Based on Eric Roberts' genlib.c and simpio.c.
12: *
13: * The latest version of this file can be found at
14: * http://cs50.net/pub/releases/cs50/cs50.c.
15: *
16: * To compile as a static library on your own system:
17: * % gcc -c -ggdb -std=c99 cs50.c -o cs50.o
18: * % ar rcs libcs50.a cs50.o
19: * % rm -f cs50.o
20: * % cp cs50.h /usr/local/include
21: * % cp libcs50.a /usr/local/lib
22: *****/
23:
24: #include <stdio.h>
25: #include <stdlib.h>
26: #include <string.h>
27:
28: #include "cs50.h"
29:
30:
31: /*
32:  * Default capacity of buffer for standard input.
33:  */
34:
35: #define CAPACITY 128
36:
37:
38: /*
39:  * char
40:  * GetChar()
41:  *
42:  * Reads a line of text from standard input and returns the equivalent
43:  * char; if text does not represent a char, user is prompted to retry.
44:  * Leading and trailing whitespace is ignored. If line can't be read,
45:  * returns CHAR_MAX.
46:  */
47:
48: char
49: GetChar()
50: {
51:     // try to get a char from user
52:     while (true)
53:     {
54:         // get line of text, returning CHAR_MAX on failure
55:         string line = GetString();
56:         if (line == NULL)
57:             return CHAR_MAX;
58:
59:         // return a char if only a char (possibly with
60:         // leading and/or trailing whitespace) was provided
61:         char c1, c2;
62:         if (sscanf(line, " %c %c", &c1, &c2) == 1)
63:         {
64:             free(line);

```

```

65:         return c1;
66:     }
67:     else
68:     {
69:         free(line);
70:         printf("Retry: ");
71:     }
72: }
73: }
74:
75:
76: /*
77:  * double
78:  * GetDouble()
79:  *
80:  * Reads a line of text from standard input and returns the equivalent
81:  * double as precisely as possible; if text does not represent a
82:  * double, user is prompted to retry. Leading and trailing whitespace
83:  * is ignored. For simplicity, overflow and underflow are not detected.
84:  * If line can't be read, returns DBL_MAX.
85:  */
86:
87: double
88: GetDouble()
89: {
90:     // try to get a double from user
91:     while (true)
92:     {
93:         // get line of text, returning DBL_MAX on failure
94:         string line = GetString();
95:         if (line == NULL)
96:             return DBL_MAX;
97:
98:         // return a double if only a double (possibly with
99:         // leading and/or trailing whitespace) was provided
100:        double d; char c;
101:        if (sscanf(line, " %lf %c", &d, &c) == 1)
102:        {
103:            free(line);
104:            return d;
105:        }
106:        else
107:        {
108:            free(line);
109:            printf("Retry: ");
110:        }
111:    }
112: }
113:
114:
115: /*
116:  * float
117:  * GetFloat()
118:  *
119:  * Reads a line of text from standard input and returns the equivalent
120:  * float as precisely as possible; if text does not represent a float,
121:  * user is prompted to retry. Leading and trailing whitespace is ignored.
122:  * For simplicity, overflow and underflow are not detected. If line can't
123:  * be read, returns FLT_MAX.
124:  */
125:
126: float
127: GetFloat()
128: {

```

```

129: // try to get a float from user
130: while (true)
131: {
132:     // get line of text, returning FLT_MAX on failure
133:     string line = GetString();
134:     if (line == NULL)
135:         return FLT_MAX;
136:
137:     // return a float if only a float (possibly with
138:     // leading and/or trailing whitespace) was provided
139:     char c; float f;
140:     if (sscanf(line, "%f %c", &f, &c) == 1)
141:     {
142:         free(line);
143:         return f;
144:     }
145:     else
146:     {
147:         free(line);
148:         printf("Retry: ");
149:     }
150: }
151: }
152:
153:
154: /*
155: * int
156: * GetInt()
157: *
158: * Reads a line of text from standard input and returns it as an
159: * int in the range of [-2^31 + 1, 2^31 - 2], if possible; if text
160: * does not represent such an int, user is prompted to retry. Leading
161: * and trailing whitespace is ignored. For simplicity, overflow is not
162: * detected. If line can't be read, returns INT_MAX.
163: */
164:
165: int
166: GetInt()
167: {
168:     // try to get an int from user
169:     while (true)
170:     {
171:         // get line of text, returning INT_MAX on failure
172:         string line = GetString();
173:         if (line == NULL)
174:             return INT_MAX;
175:
176:         // return an int if only an int (possibly with
177:         // leading and/or trailing whitespace) was provided
178:         int n; char c;
179:         if (sscanf(line, "%d %c", &n, &c) == 1)
180:         {
181:             free(line);
182:             return n;
183:         }
184:         else
185:         {
186:             free(line);
187:             printf("Retry: ");
188:         }
189:     }
190: }
191:
192:

```

```

193: /*
194: * long long
195: * GetLongLong()
196: *
197: * Reads a line of text from standard input and returns an equivalent
198: * long long in the range [-2^63 + 1, 2^63 - 2], if possible; if text
199: * does not represent such a long long, user is prompted to retry.
200: * Leading and trailing whitespace is ignored. For simplicity, overflow
201: * is not detected. If line can't be read, returns LLONG_MAX.
202: */
203:
204: long long
205: GetLongLong()
206: {
207:     // try to get a long long from user
208:     while (true)
209:     {
210:         // get line of text, returning LLONG_MAX on failure
211:         string line = GetString();
212:         if (line == NULL)
213:             return LLONG_MAX;
214:
215:         // return a long long if only a long long (possibly with
216:         // leading and/or trailing whitespace) was provided
217:         long long n; char c;
218:         if (sscanf(line, "%lld %c", &n, &c) == 1)
219:         {
220:             free(line);
221:             return n;
222:         }
223:         else
224:         {
225:             free(line);
226:             printf("Retry: ");
227:         }
228:     }
229: }
230:
231:
232: /*
233: * string
234: * GetString()
235: *
236: * Reads a line of text from standard input and returns it as a string,
237: * sans trailing newline character. (Ergo, if user inputs only "\n",
238: * returns "" not NULL.) Leading and trailing whitespace is not ignored.
239: * Returns NULL upon error or no input whatsoever (i.e., just EOF).
240: */
241:
242: string
243: GetString()
244: {
245:     // growable buffer for chars
246:     string buffer = NULL;
247:
248:     // capacity of buffer
249:     unsigned int capacity = 0;
250:
251:     // number of chars actually in buffer
252:     unsigned int n = 0;
253:
254:     // character read or EOF
255:     int c;
256:

```

```

257: // iteratively get chars from standard input
258: while ((c = fgetc(stdin)) != '\n' && c != EOF)
259: {
260:     // grow buffer if necessary
261:     if (n + 1 > capacity)
262:     {
263:         // determine new capacity: start at CAPACITY then double
264:         if (capacity == 0)
265:             capacity = CAPACITY;
266:         else if (capacity <= (UINT_MAX / 2))
267:             capacity += 2;
268:         else
269:         {
270:             free(buffer);
271:             return NULL;
272:         }
273:
274:         // extend buffer's capacity
275:         string temp = realloc(buffer, capacity * sizeof(char));
276:         if (temp == NULL)
277:         {
278:             free(buffer);
279:             return NULL;
280:         }
281:         buffer = temp;
282:     }
283:
284:     // append current character to buffer
285:     buffer[n++] = c;
286: }
287:
288: // return NULL if user provided no input
289: if (n == 0 && c == EOF)
290:     return NULL;
291:
292: // minimize buffer
293: string minimal = malloc((n + 1) * sizeof(char));
294: strncpy(minimal, buffer, n);
295: free(buffer);
296:
297: // terminate string
298: minimal[n] = '\0';
299:
300: // return string
301: return minimal;
302: }

```

```

1: /*****
2:  * cs50.h
3:  *
4:  * version 1.1.3
5:  *
6:  * Computer Science 50
7:  * Glenn Holloway
8:  * David J. Malan
9:  *
10: * Declarations for CS 50's library.
11: * Based on Eric Roberts' genlib.h and simpio.h.
12: *
13: * The latest version of this file can be found at
14: * http://cs50.net/pub/releases/cs50/cs50.h.
15: *
16: * To compile as a static library on your own system:
17: * % gcc -c -ggdb -std=c99 cs50.c -o cs50.o
18: * % ar rcs libcs50.a cs50.o
19: * % rm -f cs50.o
20: * % cp cs50.h /usr/local/include
21: * % cp libcs50.a /usr/local/lib
22: *****/
23:
24: #ifndef _CS50_H
25: #define _CS50_H
26:
27: #include <float.h>
28: #include <limits.h>
29:
30:
31: /*
32:  * bool
33:  *
34:  * Borrow the standard library's data type for Boolean variables whose
35:  * values must be (true|false).
36:  */
37:
38: #include <stdbool.h>
39:
40:
41: /*
42:  * string
43:  *
44:  * Our own data type for string variables.
45:  */
46:
47: typedef char *string;
48:
49:
50: /*
51:  * char
52:  * GetChar()
53:  *
54:  * Reads a line of text from standard input and returns the equivalent
55:  * char; if text does not represent a char, user is prompted to retry.
56:  * Leading and trailing whitespace is ignored. If line can't be read,
57:  * returns CHAR_MAX.
58:  */
59:
60: char
61: GetChar();
62:
63:
64: /*

```

```

65: * double
66: * GetDouble()
67: *
68: * Reads a line of text from standard input and returns the equivalent
69: * double as precisely as possible; if text does not represent a
70: * double, user is prompted to retry. Leading and trailing whitespace
71: * is ignored. For simplicity, overflow and underflow are not detected.
72: * If line can't be read, returns DBL_MAX.
73: */
74:
75: double
76: GetDouble();
77:
78:
79: /*
80: * float
81: * GetFloat()
82: *
83: * Reads a line of text from standard input and returns the equivalent
84: * float as precisely as possible; if text does not represent a float,
85: * user is prompted to retry. Leading and trailing whitespace is ignored.
86: * For simplicity, overflow and underflow are not detected. If line can't
87: * be read, returns FLT_MAX.
88: */
89:
90: float
91: GetFloat();
92:
93:
94: /*
95: * int
96: * GetInt()
97: *
98: * Reads a line of text from standard input and returns it as an
99: * int in the range of  $[-2^{31} + 1, 2^{31} - 2]$ , if possible; if text
100: * does not represent such an int, user is prompted to retry. Leading
101: * and trailing whitespace is ignored. For simplicity, overflow is not
102: * detected. If line can't be read, returns INT_MAX.
103: */
104:
105: int
106: GetInt();
107:
108:
109: /*
110: * long long
111: * GetLongLong()
112: *
113: * Reads a line of text from standard input and returns an equivalent
114: * long long in the range  $[-2^{63} + 1, 2^{63} - 2]$ , if possible; if text
115: * does not represent such a long long, user is prompted to retry.
116: * Leading and trailing whitespace is ignored. For simplicity, overflow
117: * is not detected. If line can't be read, returns LLONG_MAX.
118: */
119:
120: long long
121: GetLongLong();
122:
123:
124: /*
125: * string
126: * GetString()
127: *
128: * Reads a line of text from standard input and returns it as a string,

```

```

129: * sans trailing newline character. (Ergo, if user inputs only "\n",
130: * returns "" not NULL.) Leading and trailing whitespace is not ignored.
131: * Returns NULL upon error or no input whatsoever (i.e., just EOF).
132: */
133:
134: string GetString();
135:
136:
137:
138: #endif

```

```
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:     // get line of text
22:     char *s = GetString();
23:     if (s == NULL)
24:         return 1;
25:
26:     // print string, one character per line
27:     for (int i = 0, n = strlen(s); i < n; i++)
28:         printf("%c\n", *(s+i));
29:
30:     // free string
31:     free(s);
32: }
```

```
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:  * 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:     printf("Number please: ");
20:     scanf("%d", &x);
21:     printf("Thanks for the %d!\n", x);
22: }
```

```
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:     printf("String please: ");
20:     scanf("%s", buffer);
21:     printf("Thanks for the \"%s\"!\n", buffer);
22: }
```

```
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:     printf("String please: ");
20:     scanf("%s", buffer);
21:     printf("Thanks for the \"%s\"!\n", buffer);
22: }
```

```
1: /*****
2:  * structs.h
3:  *
4:  * Computer Science 50
5:  * David J. Malan
6:  *
7:  * Defines a student for structs{1,2}.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;
```

```
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 size
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: }
```

```
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 size
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: }
```

```
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: }
```