

```

1: /*****
2:  * cs50.c
3:  *
4:  * version 1.1.6
5:  *
6:  * Computer Science 50
7:  * Glenn Holloway
8:  * David J. Malan
9:  *
10: * Definitions for the CS50 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://www.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:  * Default capacity of buffer for standard input.
32:  */
33:
34: #define CAPACITY 128
35:
36:
37:
38: /*
39:  * Reads a line of text from standard input and returns the equivalent
40:  * char; if text does not represent a char, user is prompted to retry.
41:  * Leading and trailing whitespace is ignored. If line can't be read,
42:  * returns CHAR_MAX.
43:  */
44:
45: char
46: GetChar(void)
47: {
48:     // try to get a char from user
49:     while (true)
50:     {
51:         // get line of text, returning CHAR_MAX on failure
52:         string line = GetString();
53:         if (line == NULL)
54:             return CHAR_MAX;
55:
56:         // return a char if only a char (possibly with
57:         // leading and/or trailing whitespace) was provided
58:         char c1, c2;
59:         if (sscanf(line, " %c %c", &c1, &c2) == 1)
60:         {
61:             free(line);
62:             return c1;
63:         }
64:         else

```

```

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

```

```

129:         // leading and/or trailing whitespace) was provided
130:         char c; float f;
131:         if (sscanf(line, " %f %c", &f, &c) == 1)
132:         {
133:             free(line);
134:             return f;
135:         }
136:         else
137:         {
138:             free(line);
139:             printf("Retry: ");
140:         }
141:     }
142: }
143:
144: /*
145: * Reads a line of text from standard input and returns it as an
146: * int in the range of [-2^31 + 1, 2^31 - 2], if possible; if text
147: * does not represent such an int, user is prompted to retry. Leading
148: * and trailing whitespace is ignored. For simplicity, overflow is not
149: * detected. If line can't be read, returns INT_MAX.
150: */
151: int
152: GetInt(void)
153: {
154:     // try to get an int from user
155:     while (true)
156:     {
157:         // get line of text, returning INT_MAX on failure
158:         string line = GetString();
159:         if (line == NULL)
160:             return INT_MAX;
161:
162:         // return an int if only an int (possibly with
163:         // leading and/or trailing whitespace) was provided
164:         int n; char c;
165:         if (sscanf(line, " %d %c", &n, &c) == 1)
166:         {
167:             free(line);
168:             return n;
169:         }
170:         else
171:         {
172:             free(line);
173:             printf("Retry: ");
174:         }
175:     }
176: }
177:
178: /*
179: * Reads a line of text from standard input and returns an equivalent
180: * long long in the range [-2^63 + 1, 2^63 - 2], if possible; if text
181: * does not represent such a long long, user is prompted to retry.
182: * Leading and trailing whitespace is ignored. For simplicity, overflow
183: * is not detected. If line can't be read, returns LLONG_MAX.
184: */
185: long long
186: GetLongLong(void)
187: {
188:     // try to get a long long from user

```

```

193:     while (true)
194:     {
195:         // get line of text, returning LLONG_MAX on failure
196:         string line = GetString();
197:         if (line == NULL)
198:             return LLONG_MAX;
199:
200:         // return a long long if only a long long (possibly with
201:         // leading and/or trailing whitespace) was provided
202:         long long n; char c;
203:         if (sscanf(line, " %lld %c", &n, &c) == 1)
204:         {
205:             free(line);
206:             return n;
207:         }
208:         else
209:         {
210:             free(line);
211:             printf("Retry: ");
212:         }
213:     }
214: }
215:
216: /*
217: * Reads a line of text from standard input and returns it as a string,
218: * sans trailing newline character. (Ergo, if user inputs only "\n",
219: * returns "" not NULL.) Leading and trailing whitespace is not ignored.
220: * Returns NULL upon error or no input whatsoever (i.e., just EOF).
221: */
222: string
223: GetString(void)
224: {
225:     // growable buffer for chars
226:     string buffer = NULL;
227:
228:     // capacity of buffer
229:     unsigned int capacity = 0;
230:
231:     // number of chars actually in buffer
232:     unsigned int n = 0;
233:
234:     // character read or EOF
235:     int c;
236:
237:     // iteratively get chars from standard input
238:     while ((c = fgetc(stdin)) != '\n' && c != EOF)
239:     {
240:         // grow buffer if necessary
241:         if (n + 1 > capacity)
242:         {
243:             // determine new capacity: start at CAPACITY then double
244:             if (capacity == 0)
245:                 capacity = CAPACITY;
246:             else if (capacity <= (UINT_MAX / 2))
247:                 capacity *= 2;
248:             else
249:             {
250:                 free(buffer);
251:                 return NULL;
252:             }
253:         }
254:
255:         // extend buffer's capacity

```

```

257:         string temp = realloc(buffer, capacity * sizeof(char));
258:         if (temp == NULL)
259:         {
260:             free(buffer);
261:             return NULL;
262:         }
263:         buffer = temp;
264:     }
265:
266:     // append current character to buffer
267:     buffer[n++] = c;
268: }
269:
270: // return NULL if user provided no input
271: if (n == 0 && c == EOF)
272:     return NULL;
273:
274: // minimize buffer
275: string minimal = malloc((n + 1) * sizeof(char));
276: strncpy(minimal, buffer, n);
277: free(buffer);
278:
279: // terminate string
280: minimal[n] = '\0';
281:
282: // return string
283: return minimal;
284: }

```

```

1: /*****
2:  * cs50.h
3:  *
4:  * version 1.1.6
5:  *
6:  * Computer Science 50
7:  * Glenn Holloway
8:  * David J. Malan
9:  *
10: * Declarations for the CS50 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://www.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:  * Borrow the standard library's data type for Boolean variables whose
33:  * values must be (true|false).
34:  */
35:
36: #include <stdbool.h>
37:
38:
39: /*
40:  * Our own data type for string variables.
41:  */
42:
43: typedef char *string;
44:
45:
46: /*
47:  * Reads a line of text from standard input and returns the equivalent
48:  * char; if text does not represent a char, user is prompted to retry.
49:  * Leading and trailing whitespace is ignored. If line can't be read,
50:  * returns CHAR_MAX.
51:  */
52:
53: char
54: GetChar(void);
55:
56:
57: /*
58:  * Reads a line of text from standard input and returns the equivalent
59:  * double as precisely as possible; if text does not represent a
60:  * double, user is prompted to retry. Leading and trailing whitespace
61:  * is ignored. For simplicity, overflow and underflow are not detected.
62:  * If line can't be read, returns DBL_MAX.
63:  */
64:

```

```

65: double
66: GetDouble(void);
67:
68:
69: /*
70:  * Reads a line of text from standard input and returns the equivalent
71:  * float as precisely as possible; if text does not represent a float,
72:  * user is prompted to retry. Leading and trailing whitespace is ignored.
73:  * For simplicity, overflow and underflow are not detected. If line can't
74:  * be read, returns FLT_MAX.
75:  */
76:
77: float
78: GetFloat(void);
79:
80:
81: /*
82:  * Reads a line of text from standard input and returns it as an
83:  * int in the range of  $[-2^{31} + 1, 2^{31} - 2]$ , if possible; if text
84:  * does not represent such an int, user is prompted to retry. Leading
85:  * and trailing whitespace is ignored. For simplicity, overflow is not
86:  * detected. If line can't be read, returns INT_MAX.
87:  */
88:
89: int
90: GetInt(void);
91:
92:
93: /*
94:  * Reads a line of text from standard input and returns an equivalent
95:  * long long in the range  $[-2^{63} + 1, 2^{63} - 2]$ , if possible; if text
96:  * does not represent such a long long, user is prompted to retry.
97:  * Leading and trailing whitespace is ignored. For simplicity, overflow
98:  * is not detected. If line can't be read, returns LLONG_MAX.
99:  */
100:
101: long long
102: GetLongLong(void);
103:
104:
105: /*
106:  * Reads a line of text from standard input and returns it as a string,
107:  * sans trailing newline character. (Ergo, if user inputs only "\n",
108:  * returns "" not NULL.) Leading and trailing whitespace is not ignored.
109:  * Returns NULL upon error or no input whatsoever (i.e., just EOF).
110:  */
111:
112: string GetString(void);
113:
114:
115:
116: #endif

```

```

1: /*****
2:  * list1.c
3:  *
4:  * Computer Science 50
5:  * David J. Malan
6:  *
7:  * Demonstrates a linked list for numbers.
8:  *****/
9:
10:
11: #include <cs50.h>
12: #include <stdio.h>
13: #include <stdlib.h>
14: #include <unistd.h>
15:
16: #include "list1.h"
17:
18:
19: // linked list
20: node *first = NULL;
21:
22:
23: // prototypes
24: void delete(void);
25: void find(void);
26: void insert(void);
27: void traverse(void);
28:
29:
30: int
31: main(void)
32: {
33:     int c;
34:     do
35:     {
36:         // print instructions
37:         printf("\nMENU\n\n"
38:             "1 - delete\n"
39:             "2 - find\n"
40:             "3 - insert\n"
41:             "4 - traverse\n"
42:             "0 - quit\n\n");
43:
44:         // get command
45:         printf("Command: ");
46:         c = GetInt();
47:
48:         // try to execute command
49:         switch (c)
50:         {
51:             case 1: delete(); break;
52:             case 2: find(); break;
53:             case 3: insert(); break;
54:             case 4: traverse(); break;
55:         }
56:     }
57:     while (c != 0);
58:
59:     // free list before quitting
60:     node *ptr = first;
61:     while (ptr != NULL)
62:     {
63:         node *predptr = ptr;
64:         ptr = ptr->next;

```

```

65:     free(predptr);
66: }
67: return 0;
68: }
69:
70:
71: /*
72:  * Tries to delete a number.
73:  */
74:
75: void
76: delete(void)
77: {
78:     // prompt user for number
79:     printf("Number to delete: ");
80:     int n = GetInt();
81:
82:     // get list's first node
83:     node *ptr = first;
84:
85:     // try to delete number from list
86:     node *predptr = NULL;
87:     while (ptr != NULL)
88:     {
89:         // check for number
90:         if (ptr->n == n)
91:         {
92:             // delete from head
93:             if (ptr == first)
94:             {
95:                 first = ptr->next;
96:                 free(ptr);
97:             }
98:
99:             // delete from middle or tail
100:            else
101:            {
102:                predptr->next = ptr->next;
103:                free(ptr);
104:            }
105:
106:            // all done
107:            break;
108:        }
109:        else
110:        {
111:            predptr = ptr;
112:            ptr = ptr->next;
113:        }
114:    }
115:
116:    // traverse list
117:    traverse();
118: }
119:
120:
121: /*
122:  * Tries to insert a number into list.
123:  */
124:
125: void
126: insert(void)
127: {
128:     // try to instantiate node for number

```

```

129:     node *newptr = malloc(sizeof(node));
130:     if (newptr == NULL)
131:         return;
132:
133:     // initialize node
134:     printf("Number to insert: ");
135:     newptr->n = GetInt();
136:     newptr->next = NULL;
137:
138:     // check for empty list
139:     if (first == NULL)
140:         first = newptr;
141:
142:     // else check if number belongs at list's head
143:     else if (newptr->n < first->n)
144:     {
145:         newptr->next = first;
146:         first = newptr;
147:     }
148:
149:     // else try to insert number in middle or tail
150:     else
151:     {
152:         node *predptr = first;
153:         while (true)
154:         {
155:             // avoid duplicates
156:             if (predptr->n == newptr->n)
157:             {
158:                 free(newptr);
159:                 break;
160:             }
161:
162:             // check for insertion at tail
163:             else if (predptr->next == NULL)
164:             {
165:                 predptr->next = newptr;
166:                 break;
167:             }
168:
169:             // check for insertion in middle
170:             else if (predptr->next->n > newptr->n)
171:             {
172:                 newptr->next = predptr->next;
173:                 predptr->next = newptr;
174:                 break;
175:             }
176:
177:             // update pointer
178:             predptr = predptr->next;
179:         }
180:     }
181:
182:     // traverse list
183:     traverse();
184: }
185:
186:
187: /*
188:  * Tries to find a number in list.
189:  */
190:
191: void
192: find(void)

```

```

193: {
194:     // prompt user for number
195:     printf("Number to find: ");
196:     int n = GetInt();
197:
198:     // get list's first node
199:     node *ptr = first;
200:
201:     // try to find number
202:     while (ptr != NULL)
203:     {
204:         if (ptr->n == n)
205:         {
206:             printf("\nFound %d!\n", n);
207:             sleep(1);
208:             break;
209:         }
210:         ptr = ptr->next;
211:     }
212: }
213:
214:
215: /*
216:  * Traverses list, printing its numbers.
217:  */
218: void
219: traverse(void)
220: {
221:     // traverse list
222:     printf("\nLIST IS NOW: ");
223:     node *ptr = first;
224:     while (ptr != NULL)
225:     {
226:         printf("%d ", ptr->n);
227:         ptr = ptr->next;
228:     }
229:
230:     // flush standard output since we haven't outputted any newlines yet
231:     fflush(stdout);
232:
233:     // pause before continuing
234:     sleep(1);
235:     printf("\n\n");
236: }

```

```

1: /*****
2:  * list1.h
3:  *
4:  * Computer Science 50
5:  * David J. Malan
6:  *
7:  * Defines a node for a linked list of integers.
8:  *****/
9:
10:
11: typedef struct node
12: {
13:     int n;
14:     struct node *next;
15: }
16: node;

```

```

1: /*****
2:  * list2.c
3:  *
4:  * Computer Science 50
5:  * David J. Malan
6:  *
7:  * Demonstrates a linked list for students.
8:  *****/
9:
10:
11: #include <cs50.h>
12: #include <stdio.h>
13: #include <stdlib.h>
14: #include <unistd.h>
15:
16: #include "list2.h"
17:
18:
19: // linked list
20: node *first = NULL;
21:
22:
23: // prototypes
24: void delete(void);
25: void find(void);
26: void insert(void);
27: void traverse(void);
28:
29:
30: int
31: main(void)
32: {
33:     int c;
34:     do
35:     {
36:         // print instructions
37:         printf("\nMENU\n\n");
38:         "1 - delete\n";
39:         "2 - find\n";
40:         "3 - insert\n";
41:         "4 - traverse\n";
42:         "0 - quit\n\n");
43:
44:         // get command
45:         printf("Command: ");
46:         c = GetInt();
47:
48:         // try to execute command
49:         switch (c)
50:         {
51:             case 1: delete(); break;
52:             case 2: find(); break;
53:             case 3: insert(); break;
54:             case 4: traverse(); break;
55:         }
56:     }
57:     while (c != 0);
58:
59:     // free list before quitting
60:     node *ptr = first;
61:     while (ptr != NULL)
62:     {
63:         node *predptr = ptr;
64:         ptr = ptr->next;

```

```

65:         free(predptr);
66:     }
67:     return 0;
68: }
69:
70:
71: /*
72:  * Tries to delete a student.
73:  */
74:
75: void
76: delete(void)
77: {
78:     // prompt user for ID
79:     printf("ID to delete: ");
80:     int n = GetInt();
81:
82:     // get list's first node
83:     node *ptr = first;
84:
85:     // try to delete student from list
86:     node *predptr = NULL;
87:     while (ptr != NULL)
88:     {
89:         // check for ID
90:         if (ptr->student->id == n)
91:         {
92:             // delete from head
93:             if (ptr == first)
94:             {
95:                 first = ptr->next;
96:                 free(ptr->student->name);
97:                 free(ptr->student->house);
98:                 free(ptr->student);
99:                 free(ptr);
100:             }
101:
102:             // delete from middle or tail
103:             else
104:             {
105:                 predptr->next = ptr->next;
106:                 if (ptr->student->name != NULL)
107:                     free(ptr->student->name);
108:                 if (ptr->student->house != NULL)
109:                     free(ptr->student->house);
110:                 free(ptr->student);
111:                 free(ptr);
112:             }
113:
114:             // all done
115:             break;
116:         }
117:         else
118:         {
119:             predptr = ptr;
120:             ptr = ptr->next;
121:         }
122:     }
123:
124:     // traverse list
125:     traverse();
126: }
127:
128:

```

```

129: /*
130:  * Tries to insert a student into list.
131:  */
132:
133: void
134: insert(void)
135: {
136:     // try to instantiate node for student
137:     node *newptr = malloc(sizeof(node));
138:     if (newptr == NULL)
139:         return;
140:
141:     // initialize node
142:     newptr->next = NULL;
143:
144:     // try to instantiate student
145:     newptr->student = malloc(sizeof(student));
146:     if (newptr->student == NULL)
147:     {
148:         free(newptr);
149:         return;
150:     }
151:
152:     // try to initialize student
153:     printf("Student's ID: ");
154:     newptr->student->id = GetInt();
155:     printf("Student's name: ");
156:     newptr->student->name = GetString();
157:     printf("Student's house: ");
158:     newptr->student->house = GetString();
159:     if (newptr->student->name == NULL || newptr->student->house == NULL)
160:     {
161:         if (newptr->student->name != NULL)
162:             free(newptr->student->name);
163:         if (newptr->student->house != NULL)
164:             free(newptr->student->house);
165:         free(newptr->student);
166:         free(newptr);
167:         return;
168:     }
169:
170:     // check for empty list
171:     if (first == NULL)
172:         first = newptr;
173:
174:     // else check if student belongs at list's head
175:     else if (newptr->student->id < first->student->id)
176:     {
177:         newptr->next = first;
178:         first = newptr;
179:     }
180:
181:     // else try to insert student in middle or tail
182:     else
183:     {
184:         node *predptr = first;
185:         while (true)
186:         {
187:             // avoid duplicates
188:             if (predptr->student->id == newptr->student->id)
189:             {
190:                 free(newptr->student->name);
191:                 free(newptr->student->house);
192:                 free(newptr->student);

```

```

193:         free(newptr);
194:         break;
195:     }
196:
197:     // check for insertion at tail
198:     else if (predptr->next == NULL)
199:     {
200:         predptr->next = newptr;
201:         break;
202:     }
203:
204:     // check for insertion in middle
205:     else if (predptr->next->student->id > newptr->student->id)
206:     {
207:         newptr->next = predptr->next;
208:         predptr->next = newptr;
209:         break;
210:     }
211:
212:     // update pointer
213:     predptr = predptr->next;
214: }
215: }
216:
217: // traverse list
218: traverse();
219: }
220:
221:
222: /*
223:  * Tries to find a number in list.
224:  */
225:
226: void
227: find(void)
228: {
229:     // prompt user for ID
230:     printf("ID to find: ");
231:     int id = GetInt();
232:
233:     // get list's first node
234:     node *ptr = first;
235:
236:     // try to find student
237:     while (ptr != NULL)
238:     {
239:         if (ptr->student->id == id)
240:         {
241:             printf("\nFound %s of %s (%d)!\n",
242:                 ptr->student->name, ptr->student->house, id);
243:             sleep(1);
244:             break;
245:         }
246:         ptr = ptr->next;
247:     }
248: }
249:
250:
251: /*
252:  * Traverses list, printing its numbers.
253:  */
254:
255: void
256: traverse(void)

```


list2.c

5/5

lectures/5/src/

```
257: {
258:     // traverse list
259:     printf("\nLIST IS NOW: ");
260:     node *ptr = first;
261:     while (ptr != NULL)
262:     {
263:         printf("%s of %s (%d) ",
264:             ptr->student->name, ptr->student->house, ptr->student->id);
265:         ptr = ptr->next;
266:     }
267:
268:     // flush standard output since we haven't outputted any newlines yet
269:     fflush(stdout);
270:
271:     // pause before continuing
272:     sleep(1);
273:     printf("\n\n");
274: }
```

list2.h

1/1

lectures/5/src/

```
1: /*****
2:  * list2.h
3:  *
4:  * Computer Science 50
5:  * David J. Malan
6:  *
7:  * Defines structures for students and linked lists thereof.
8:  *****/
9:
10:
11: typedef struct
12: {
13:     int id;
14:     char *name;
15:     char *house;
16: }
17: student;
18:
19:
20: typedef struct node
21: {
22:     student *student;
23:     struct node *next;
24: }
25: node;
```

structs.h

1/1

lectures/5/src/

```
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;
```

structs1.c

1/1

lectures/5/src/

```
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(void)
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(void)
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: }
```