```
 1: #include <stdio.h>
 2:
 3: int main(int argc, char *argv[]) {
 4:   if(argc != 3)
 5:     return 1;
 6:
 7:   FILE *src = fopen(argv[1], "r");
 8:   FILE *dest = fopen(argv[2], "w");
 9:
10:   char ch;
11:   while((ch = fgetc(src)) != EOF)
12:     fputc(ch, dest);
13:
14:   fclose(src);
15:   fclose(dest);
16:
17:   return 0;
18: }
```

```
 1: #include <stdio.h>
 2:
 3: /* Constants */
 4: #define NUM_MONTHS 12
 5:
 6: /* Type definitions */
 7: typedef enum { FALSE, TRUE } doug_bool;
 8:
 9: typedef enum { JAN = 1, FEB, MAR, APR, MAY, JUN,
10:                JUL, AUG, SEP, OCT, NOV, DEC } month;
11:
12: /* next_month() calculates the next month! */
13: month next_month(month m) {
14:   return m % NUM_MONTHS + 1;
15: }
16:
17:
18: int main() {
19:   doug_bool x = TRUE;
20:   if(x)
21:     printf("x is true!\n");
22:
23:   doug_bool y = FALSE;
24:   if(y)
25:     printf("this shouldn't print!\n");
26:
27:   month october = OCT;
28:   month december = DEC;
29:
30:   printf("October is the %dth month of the year!\n", october);
31:   printf("November is the %dth month of the year!\n", next_month(october));
32:   printf("Then comes December, and after that it's the %dst month again!\n",
33:          next_month(december));
34:
35:   return 0;
36: }
```

```
 1: COPY TESTING
 2:
 3: 1 999999999
 4: 22 88888888
 5: 333 7777777
 6: 4444 666666
 7: 55555 55555
 8: 666666 4444
 9: 7777777 333
10: 88888888 22
11: 999999999 1
12:
13: ABCDEFGHIJKLMNOPQRSTUVWXYZ
14: abcdefghijklmnopqrstuvwxyz
15: ‘~!@#$%^&*()-_+=[]{}\|
16: :;’",<.>/?
```

```
 1: // CS50
 2: #include <cs50.h>
 3:
 4: // structure definition
 5: typedef struct _sllist {
 6:   int data;
 7:   struct _sllist *next;
 8: } sllist;
 9:
10: // print a linked list iteratively
11: void printList_I(sllist *head);
12:
13: // print a linked list recursively
14: void printList_R(sllist *head);
15:
16: // create a linked list
17: sllist *llist_create(int val);
18:
19: // add to the end of a linked list
20: sllist *llist_append(sllist *head, int val);
21:
22: // add after some element in the middle of a linked list
23: bool llist_insert_after(sllist *x, int val);
24:
25: // destroy a list
26: void llist_destroy_list(sllist *head);
27:
28: // find an element in a linked list
29: sllist *llist_find(sllist *head, int val);
```

```
 1: #include <stdio.h>
 2: #include <stdlib.h>
 3: #include "sllist.h"
 4:
 5: int main() {
 6:   sllist first;
 7:   sllist second;
 8:   sllist *third = malloc(sizeof(sllist));
 9:
10:   first.data = 5;
11:   second.data = 7;
12:   third->data = 9;
13:
14:   first.next = &second;
15:   second.next = third;
16:   third->next = NULL;
17:
18:   printList_I(&first);
19:   printf("\n");
20:   printList_R(&first);
21:   printf("\n");
22:
23:   free(third);
24:   return 0;
25: }
26:
27: void printList_I(sllist *head) {
28:   while(head != NULL) {
29:     printf("%d ", head->data);
30:     head = head->next;
31:   }
32:   return;
33: }
34:
35: void printList_R(sllist *head) {
36:   if(head == NULL)
37:     return;
38:   else {
39:     printf("%d ", head->data);
40:     printList_R(head->next);
41:   }
42:   return;
43: }
```

```
 1: #include "sllist.h"
 2: #include <stdlib.h>
 3: #include <stdio.h>
 4:
 5: int main() {
 6:   sllist *list_head = llist_create(1);
 7:
 8:   for(int i = 2; i <= 10; i++)
 9:     list_head = llist_append(list_head, i);
10:
11:   printList_R(list_head);
12:   printf("\n");
13:
14:   sllist *insert_node = llist_find(list_head, 5);
15:   if(llist_insert_after(insert_node, 11)) {
16:     printList_R(list_head);
17:     printf("\n");
18:   }
19:   else
20:     printf("MAJOR ERROR!\n");
21:
22:   llist_destroy_list(list_head);
23:
24:   return 0;
25: }
26:
27: sllist *llist_find(sllist *head, int val) {
28:   while(head != NULL) {
29:     if(head->data == val)
30:       return head;
31:     head = head->next;
32:   }
33:
34:   return head;
35: }
36:
37: sllist *llist_create(int val) {
38:   sllist *head = malloc(sizeof(sllist));
39:   if(head == NULL) {
40:     printf("Error! malloc() failure!\n");
41:     exit(1);
42:   }
43:
44:   head->data = val;
45:
46:   return head;
47: }
48:
49: bool llist_insert_after(sllist *x, int val) {
50:   sllist *new_ele = malloc(sizeof(sllist));
51:   if(new_ele == NULL)
52:     return false;
53:
54:   new_ele->data = val;
55:
56:   new_ele->next = x->next;
57:   x->next = new_ele;
58:
59:   return true;
60: }
61:
62: sllist *llist_append(sllist *head, int val) {
63:   sllist *trav = head;
64:   while(trav->next != NULL)
```

```
65:     trav = trav->next;
66:
67:   trav->next = malloc(sizeof(sllist));
68:   if(trav->next == NULL) {
69:     printf("Failure to append node to list!\n");
70:     return head;
71:   }
72:
73:   trav->next->data = val;
74:   trav->next->next = NULL;
75:
76:   return head;
77: }
78:
79: void llist_destroy_list(sllist *head) {
80:   if(head == NULL)
81:     return;
82:   else {
83:     llist_destroy_list(head->next);
84:     free(head);
85:   }
86:   return;
87: }
88:
89: void printList_R(sllist *head) {
90:   if(head == NULL)
91:     return;
92:   else {
93:     printf("%d ", head->data);
94:     printList_R(head->next);
95:   }
96:   return;
97: }
```

```
 1: /**************************************
 2:  * struct.c
 3:  * Doug Lloyd
 4:  * October 3, 2010
 5:  *
 6:  * Fun with structs
 7:  **************************************/
 8:
 9: /* Header files */
10: #include <stdio.h>
11: #include <cs50.h>
12: #include <unistd.h>
13:
14: /* Structure Declarations */
15: struct cat_t {
16:   string name;
17:   int age;
18:   char gender;
19: };
20:
21: /* Function Declarations */
22: struct cat_t makeCat(string n, int a, char g);
23: void printCat(struct cat_t c);
24:
25: /* Function Definitions */
26: int main() {
27:
28:   // Get some info
29:   printf("What is your cat's name? ");
30:   string name = GetString();
31:   char gender;
32:   do {
33:     printf("And is it a male (M) or a female (F)? ");
34:     gender = GetChar();
35:   } while(gender != 'M' && gender != 'F');
36:   string prompt = (gender == 'M') ? "he" : "she";
37:   printf("Lastly, how old is %s? ", prompt);
38:   int age = GetInt();
39:
40:   printf("Thanks. I'll make a record for your cat now\n");
41:   sleep(1);
42:   printf("Making record...\n");
43:   struct cat_t mycat = makeCat(name, age, gender);
44:   sleep(1);
45:   printf("Record complete!\n");
46:   printCat(mycat);
47:   return 0;
48: }
49:
50: struct cat_t makeCat(string n, int a, char g) {
51:   struct cat_t xcat;
52:   xcat.name = n;
53:   xcat.age = a;
54:   xcat.gender = g;
55:   return xcat;
56: }
57:
58: void printCat(struct cat_t c) {
59:   printf("\nName: %s", c.name);
60:   printf("\nAge: %d", c.age);
61:   printf("\nGender: %c\n", c.gender);
62:   return;
63: }
64:
```

```
 1: /*************************************
 2:  * structdma.c
 3:  * Doug Lloyd
 4:  * October 3, 2010
 5:  *
 6:  * Fun with dynamically-allocated
 7:  * pointers to structs
 8:  *************************************/
 9:
10: /* Header files */
11: #include <stdio.h>
12: #include <cs50.h>
13: #include <unistd.h>
14: #include <stdlib.h>
15:
16: /* Structure Declarations */
17: struct cat_t {
18:   string name;
19:   int age;
20:   char gender;
21: };
22:
23: /* Function Declarations */
24: void makeCat(struct cat_t *xcat, string n, int a, char g);
25: void printCat(struct cat_t *c);
26:
27: /* Function Definitions */
28: int main() {
29:
30:   // Get some info
31:   printf("What is your cat's name? ");
32:   string name = GetString();
33:   char gender;
34:   do {
35:     printf("And is it a male (M) or a female (F)? ");
36:     gender = GetChar();
37:   } while(gender != 'M' && gender != 'F');
38:   string prompt = (gender == 'M') ? "he" : "she";
39:   printf("Lastly, how old is %s? ", prompt);
40:   int age = GetInt();
41:
42:   printf("Thanks. I'll make a record for your cat now\n");
43:   sleep(1);
44:   printf("Making record...\n");
45:   struct cat_t *mycat = malloc(sizeof(struct cat_t));
46:   makeCat(mycat, name, age, gender);
47:   sleep(1);
48:   printf("Record complete!\n");
49:   printCat(mycat);
50:   free(mycat);
51:   return 0;
52: }
53:
54: void makeCat(struct cat_t *xcat, string n, int a, char g) {
55:   xcat->name = n;
56:   xcat->age = a;
57:   xcat->gender = g;
58:   return;
59: }
60:
61: void printCat(struct cat_t *c) {
62:   printf("\nName: %s", c->name);
63:   printf("\nAge: %d", c->age);
64:   printf("\nGender: %c\n", c->gender);
```

```
65:   return;
66: }
67:
```

```
 1: #include <stdio.h>
 2: #include <stdlib.h>
 3:
 4: int main() {
 5:   FILE *testing = fopen("test.txt", "w");
 6:   if(testing == NULL)
 7:     return 1;
 8:   char *hw = "Hello, world!\n";
 9:
10:   fputs("foobar\n", testing);
11:   fwrite(hw, 1, 14, testing);
12:   fclose(testing);
13:   return 0;
14: }
```