```
1   /*
2    ------------------------------------------------------------------------
3    PROGRAM NAME: Lab 3 - C Bit Operations
4    PROGRAMMER:   Samuel Jentsch ACCOUNT ID: cs214114
5    CLASS:        CSC 214, Spring 2014
6    INSTRUCTOR:   Dr. Strader
7    DATE STARTED: February 11, 2014
8    DUE DATE:     February 12, 2014
9    REFERENCES:   Computer Organization and Architecture by Null and Lobor
10   Beginning C by Ivor Horton
11   Dr. Strader: assignment information sheet
12
13   PROGRAM PURPOSE:
14   Accept arguments from the command line containing commands and hex
15   values. Parse these arguments and values and manipulate the hex values
16   using bitwise operators.
17
18   VARIABLES/CONSTANTS:
19   kBIT_NUMBER = 32. Bits in the data type being used to store hex input.
20
21   METHODS:
22   Prints the individual bits in i.
23   void printBits(unsigned long i);
24
25   Gets the union (|) of set a and set b.
26   set getUnion(set a, set b);
27
28   gets the intersection (&) of set a and set b.
29   set intersection(set a, set b);
30
31   gets the complement (~) of set a.
32   set complement(set a);
33
34   Rotates the bits in x n positions.
35   unsigned long rotate(unsigned long x, int n);
36
37   Finds the number of bits set to 1 in i.
38   int numberOfBitsSet(unsigned long i);
39
40   Uses the number of bits set to 1 in i to shift all 1s to the left.
41   unsigned long shiftAllLeft(unsigned long i);

42   FILES USED:
43    NONE
44   ------------------------------------------------------------------------
45   */

46  #include <stdio.h>
47  #include "stdlib.h"
48  #include "string.h"

49  #define kBIT_NUMBER 32


50  typedef unsigned long set;
51  const set empty = 0x0;

52  /***Method Stubs***/
```

```
53  void printBits(unsigned long i);
54  set getUnion(set a, set b);
55  set intersection(set a, set b);
56  set complement(set a);
57  unsigned long rotate(unsigned long x, int n);
58  int numberOfBitsSet(unsigned long i);
59  unsigned long shiftAllLeft(unsigned long i);
60  /******************/

61  int main(int argc, const char * argv[])
62  {
63      if (argc <= 1) {
64          printf("Not enough arguments passed.\n"
65                 "Try -u, -i, -c, -r, -s, or -m.");
66          return 1;
67      }
68
69      //Holds the numeric value of the string input passed to the
70      //program.
71      unsigned long l1;
72      unsigned long l2;
73
74      //argv[1] is the command argument.
75      const char * command = argv[1];
76      if (strcmp(command, "-p") == 0) {
77          //Print
78          if (argc == 3) {
79              l1 = strtoul(argv[2], 0, 16);
80              printBits(l1);
81              printf("\n");
82          } else {
83              printf("Incorrect number of arguments for %s. Format: "
84                     "%s HEX\n", command, command);
85          }
86      } else if(strcmp(command, "-u") == 0) {
87          //Union
88          if (argc == 4) {
89              l1 = strtoul(argv[2], 0, 16);
90              l2 = strtoul(argv[3], 0, 16);
91              printf("Union of: ");
92              printBits(l1);
93              printf(" and ");
94              printBits(l2);
95              printf("is: \n");
96              printBits(getUnion(l1, l2));
97              printf("\n");
98          } else {
99              printf("Incorrect number of arguments for %s. Format: "
100                    "%s HEX HEX\n", command, command);
101         }
102     } else if(strcmp(command, "-i") == 0) {
103         //Intersection
104         if (argc == 4) {
105             l1 = strtoul(argv[2], 0, 16);
106             l2 = strtoul(argv[3], 0, 16);
107             printf("Intersection of: ");
108             printBits(l1);
109             printf(" and ");
```

```
110             printBits(l2);
111             printf("is: \n");
112             printBits(intersection(l1, l2));
113             printf("\n");
114         } else {
115             printf("Incorrect number of arguments for %s. Format: "
116                     "%s HEX HEX\n", command, command);
117         }
118     } else if(strcmp(command, "-c") == 0) {
119         //Complement
120         if (argc == 3) {
121             l1 = strtoul(argv[2], 0, 16);
122             printf("Complement of: ");
123             printBits(l1);
124             printf(" is: \n");
125             printBits(complement(l1));
126             printf("\n");
127         } else {
128             printf("Incorrect number of arguments for %s. Format: "
129                     "%s HEX\n", command, command);
130         }
131     } else if(strcmp(command, "-r") == 0) {
132         //Rotate Right
133         if (argc == 4) {
134             l1 = strtoul(argv[2], 0, 16);
135             int shiftAmount = (int)strtol(argv[3], 0, 10);
136             printf("Rotation of ");
137             printBits(l1);
138             printf(" %d positions is: \n", shiftAmount);
139             printBits(rotate(l1, shiftAmount));
140             printf("\n");
141         } else {
142             printf("Incorrect number of arguments for %s. Format: "
143                     "%s HEX INT_SHIFT_AMOUNT\n", command, command);
144         }
145     } else if(strcmp(command, "-s") == 0) {
146         //Number of Set Bits
147         if (argc == 3) {
148             l1 = strtoul(argv[2], 0, 16);
149             printf("Number of bits set in ");
150             printBits(l1);
151             printf(" is %d\n", numberOfBitsSet(l1));
152         } else {
153             printf("Incorrect number of arguments for %s. Format: "
154                     "%s HEX\n", command, command);
155         }
156     } else if(strcmp(command, "-m") == 0) {
157         //Shift all bits left
158         if (argc == 3) {
159             l1 = strtoul(argv[2], 0, 16);
160             printf("All 1s shifted left in ");
161             printBits(l1);
162             printf(" is: \n");
163             printBits(shiftAllLeft(l1));
164             printf("\n");
165         } else {
166             printf("Incorrect number of arguments for %s. Format: "
167                     "%s HEX\n", command, command);
```

```
168         }
169     } else {
170         printf("%s is an unsupported command.\n"
171                "Try -u, -i, -c, -r, -s, or -m.\n", command);
172     }
173
174     return 0;
175 }



176 void printBits(unsigned long i) {
177     //-------------------------------------------------------------------------//
178     //Prints the string representation (0 or 1) of each bit in the unsigned long
179     //i. Each bit is compared to 1 using bitwise & and a 1 is printed if bit being
180     //checked is a 1, and 0 is printed if not.
181     //Precondition: unsigned long i passed as parameter.
182     //Postcondition: The string representation of each bit in i is printed.
183     //-------------------------------------------------------------------------//
184
185     int bitNumber;
186     int j;
187
188     for (bitNumber = kBIT_NUMBER - 1, j = 1; bitNumber >= 0; bitNumber--, j++) {
189         if (i & (1 << bitNumber)) {
190             printf("1");
191         } else {
192             printf("0");
193         }
194
195         if (j == 4) {
196             printf(" ");
197             j = 0;
198         }
199     }
200 }

201 set getUnion(set a, set b) {
202     //-------------------------------------------------------------------------//
203     //Compares set a to set b using bitwise or (|). The union of a and b are the
204     //values that are present in a, or are present in b. These values are found
205     //using |.
206     //Precondition: set (unsigned long) a and set (unsigned long) b passed as
207     //parameters.
208     //Postcondition: unionSet is set to equal (a | b). (a | b) will set unionSet to
209     //the bit values of 1 present in set a, and the bit values of 1 present in set
210     //b, the union of the sets, and returns unionSet.
211     //-------------------------------------------------------------------------//
212
213     set unionSet = empty;
214
215     unionSet = a | b;
216
217     return unionSet;
218 }

219 set intersection(set a, set b) {
220     //-------------------------------------------------------------------------//
```

```
221     //Compares set a to set b using bitwise and (&). The intersection of a and b
222     //are the values that are present in a and present in b. These values are found
223     //using &.
224     //Precondition: set (unsigned long) a and set (unsigned long) b passed as
225     //parameters.
226     //Postcondition: intersectionSet is set to equal (a & b). (a & b) will set
227     //intersectionSet to the bit values of 1 that are present in both set a and set
228     //b, the intersection of the sets, and returns intersectionSet.
229     //-------------------------------------------------------------------------//
230
231     set intersectionSet = empty;
232
233     intersectionSet = a & b;
234
235     return intersectionSet;
236 }

237 set complement(set a) {
238     //-------------------------------------------------------------------------//
239     //Complements set a using the complement (~) bitwise operator.
240     //Precondition: set (unsigned long) a passed as parameter.
241     //Postcondition: complementSet is set to equal ~a, the complement of set a,
242     //and returned.
243     //-------------------------------------------------------------------------//
244
245     set complementSet = empty;
246
247     complementSet = ~a;
248
249     return complementSet;
250 }

251 unsigned long rotate(unsigned long x, int n) {
252     //-------------------------------------------------------------------------//
253     //I found the method for rotation from a Wikipedia article on circular rotation.
254     //http://en.wikipedia.org/wiki/Circular_shift.
255     //I then spent time figuring out exactly how and why it works, and include an
256     //example of it's operation.
257     //Rotates the bits of unsigned long x right the amount of positions specified
258     //by n. The bitwise operators used find the intersection of x shifted n
259     //positions right and x shifted (total bits in x - n) positions left. This
260     //method works because any bits that are "pushed" off of the end of x when it
261     //is shifted n bits right, will "wrap" around to the left side of x when it is
262     //shifted (total bits in x - n) positions left. Taking the intersection of these
263     //two sets gives the rotated value.
264     //Precondition: unsigned long x (number to rotate), int n (positions to rotate)
265     //passed as parameters.
266     //Postcondition: The bits in x are rotated right n positions and returned.
267     //-------------------------------------------------------------------------//
268
269     //Example:
270     //n is 4;
271     //0000 0000 0000 0000 0000 1111 1111 is x
272     //0000 0000 0000 0000 0000 0000 1111 x shifted right 4 positions.
273     //1111 0000 0000 0000 0000 0000 0000 x shifted left 28 positions.
274     //1111 0000 0000 0000 0000 0000 1111 is the intersection of x shifted right 4
275     //positions and x shifted left 28 positions.
276
```

```
277     x = (x >> n) | (x << (kBIT_NUMBER - n));
278
279     return x;
280 }

281 int numberOfBitsSet(unsigned long i) {
282     //-------------------------------------------------------------------------//
283     //Finds the number of bits set to 1 in i. Operates by comparing each bit in i
284     //to 1 using the bitwise and (&) operator. If a bit is 1, a count is
285     //incremented.
286     //Precondition: unsigned long i passed as parameter.
287     //Postcondition: The number of bits set to 1 in i is returned.
288     //-------------------------------------------------------------------------//

289     int bitsSet = 0;
290     int bitNumber;
291     for (bitNumber = kBIT_NUMBER - 1; bitNumber >= 0; bitNumber--) {
292         if (i & (1 << bitNumber)) {
293             bitsSet++;
294         }
295     }
296
297     return bitsSet;
298 }

299 unsigned long shiftAllLeft(unsigned long i) {
300     //-------------------------------------------------------------------------//
301     //Shifts all bits set to 1 in i to the left. Finds the number of bits set
302     //to 1 in i using numberOfBitsSet() and then creates a new unsigned long
303     //with its leftmost bits set to 1s. The number of bits set to 1 is
304     //determined by the number of 1s in i.
305     //Precondition: unsigned long i passed as parameter.
306     //Postcondition: a new unsigned long is created with its leftmost bits set to 1.
307     //The number of bits set to one is equal to the number of bits set to 1 in i.
308     //-------------------------------------------------------------------------//

309     unsigned long shiftedLeft = 0;
310     int bitsSet = numberOfBitsSet(i);
311     int bitNumber;
312
313     for (bitNumber = kBIT_NUMBER - 1; (bitsSet > 0 && bitNumber >= 0); bitNumber--, bitsSet--) {
314         shiftedLeft = shiftedLeftl(1 << bitNumber);
315     }

316     return shiftedLeft;
317 }
```