

2018 시스템 프로그래밍
- Lab 03 -

제출일자	2018.10.10
분 반	00
이 름	김민기
학 번	201502023

* datalab-handout.tar를 압축해제하고 bits.c 의 함수를 위주로 작성.
(소스코드 캡처 및 설명)

1. bitAnd

```
bits.c (~/datalab-handout) - VIM
169 * bitAnd - x&y using only ~ and |
170 *   Example: bitAnd(6, 5) = 4
171 *   Legal ops: ~ |
172 *   Max ops: 8
173 *   Rating: 1
174 */
175 int bitAnd(int x, int y) {
176     return ~(~x | ~y);
177 }
178 /*
179 * getByte - Extract byte n from word x
180 *   Bytes numbered from 0 (LSB) to 3 (MSB)
181 *   Examples: getByte(0x12345678,1) = 0x56
~/datalab-handout/bits.c [utf-8,unix][c] 2,181/285 61%
```

- 드모르간 법칙을 이용했다.
- $((A \cup B)^c = A^c \cap B^c)$

2. getByte

```
bits.c (~/datalab-handout) - VIM
177 }
178 /*
179 * getByte - Extract byte n from word x
180 *   Bytes numbered from 0 (LSB) to 3 (MSB)
181 *   Examples: getByte(0x12345678,1) = 0x56
182 *   Legal ops: ! ~ & ^ | + << >>
183 *   Max ops: 6
184 *   Rating: 2
185 */
186 int getByte(int x, int n) {
187     int result = 0;
188     n = n << 3;
189     result = x >> n;
190     result = result & 0xff;
191     return result;
192 }
193 /*
194 * logicalShift - shift x to the right by n, using a logical shift
195 *   Can assume that 0 <= n <= 31
~/datalab-handout/bits.c [utf-8,unix][c] 2,179/285 66%
```

- $n = n \ll 3$ 을 이용해 왼쪽으로 3칸을 이동(8비트의 곱)하고 $result = x \gg n$ 을 이용해 원하는 바이트를 마지막 8비트로 이동했다. 마지막으로 $result \& 0xff$ 를 통해 값을 추출하였다.

3. logicalShift

```
bits.c (~/datalab-handout) - VIM
194 /*
195  * logicalShift - shift x to the right by n, using a logical shift
196  *   Can assume that 0 <= n <= 31
197  *   Examples: logicalShift(0x87654321,4) = 0x08765432
198  *   Legal ops: ! ~ & ^ | + << >>
199  *   Max ops: 20
200  *   Rating: 3
201  */
202 int logicalShift(int x, int n) {
203     int mask = ( (1 << 31) >> n) << 1;
204
205     return (x >> n) & ~mask;
206 }
207 /*
208  * bitCount - returns count of number of 1's in word
209  *   Examples: bitCount(5) = 2, bitCount(7) = 3
210  *   Legal ops: ! ~ & ^ | + << >>
~/datalab-handout/bits.c [utf-8,unix][c] 4,210/320 63%
```

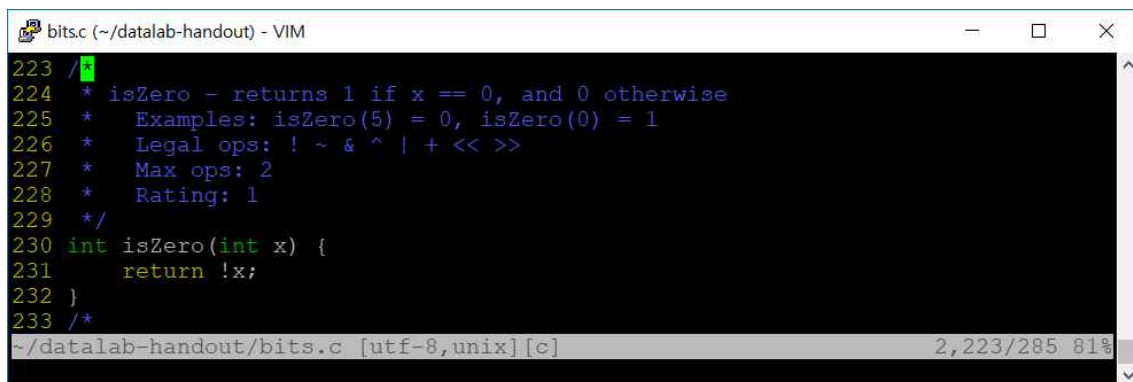
- 1 << 31을 통해 32비트 수로 변환하고, >> n 후에 << 1을 한다.
이 값에 not을 취하고 x >> n 값과 &연산을 수행하여 반환한다.

4. bitCount

```
bits.c (~/datalab-handout) - VIM
211 Examples: bitCount(5) = 2, bitCount(7) = 3
212 * Legal ops: ! ~ & ^ | + << >>
213 * Max ops: 40
214 * Rating: 4
215 */
216 int bitCount(int x) {
217     int sum = 0;
218     int total = 0;
219     int num = 0x11 + (0x11 << 8) + (0x11 << 16) + (0x11 << 24);
220     int sx = x;
221
222     sum += (num & sx);
223     sx = sx >> 1;
224     sum += (num & sx);
225     sx = sx >> 1;
226     sum += (num & sx);
227     sx = sx >> 1;
228     sum += (num & sx);
229
230     total = sum & 15;
231
232     sum = sum >> 4;
233     total += (sum & 15);
234     sum = sum >> 4;
235     total += (sum & 15);
236     sum = sum >> 4;
237     total += (sum & 15);
238     sum = sum >> 4;
239     total += (sum & 15);
240     sum = sum >> 4;
241     total += (sum & 15);
242     sum = sum >> 4;
243     total += (sum & 15);
244     sum = sum >> 4;
245     total += (sum & 15);
246
247     return total;
248 }
249 // #include "bang.c"
~/datalab-handout/bits.c [utf-8,unix][c] 2,211/322 74%
```

- num에 0x11111111을 만들어준다.
- 8개의 부분으로 나누어준다. (4비트). (부분의 첫 번째 비트합계....4번째 비트 합계까지)
- 첫 번째 부분(가장 오른쪽 부분)이 계산된다.
- $\text{sum} \gg 4$ 를 이용해 두 번째 부분에서 비트 수를 얻을 수 있다.
-8번째 부분까지 반복하면 total을 반환한다.

5. isZero



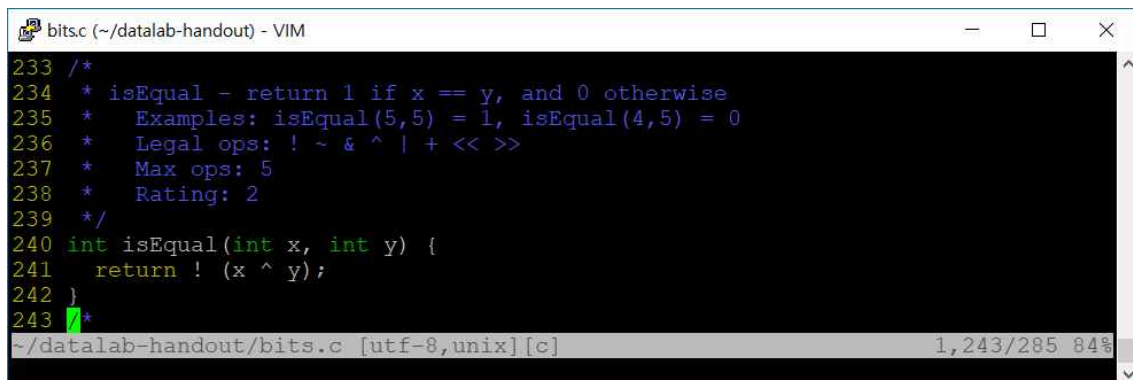
```

223 /*
224  * isZero - returns 1 if x == 0, and 0 otherwise
225  *   Examples: isZero(5) = 0, isZero(0) = 1
226  *   Legal ops: ! ~ & ^ | + << >>
227  *   Max ops: 2
228  *   Rating: 1
229  */
230 int isZero(int x) {
231     return !x;
232 }
233 */
~/datalab-handout/bits.c [utf-8,unix][c] 2,223/285 81%

```

- x의 값이 0이면 1을 반환하고, x의 값이 0이 아니면 0을 반환한다.

6. isEqual



```

233 /*
234  * isEqual - return 1 if x == y, and 0 otherwise
235  *   Examples: isEqual(5,5) = 1, isEqual(4,5) = 0
236  *   Legal ops: ! ~ & ^ | + << >>
237  *   Max ops: 5
238  *   Rating: 2
239  */
240 int isEqual(int x, int y) {
241     return ! (x ^ y);
242 }
243 */
~/datalab-handout/bits.c [utf-8,unix][c] 1,243/285 84%

```

- $1 \wedge 0 = 1$, $1 \wedge 1 = 0$ 이므로, 두 수가 같으면 0이 나와서 !를 취해서 반환한다.

7. fitsBits

```
bits.c + (~/datalab-handout) - VIM
243 /* fitsBits - return 1 if x can be represented as an
244 *    n-bit, two's complement integer.
245 *    1 <= n <= 32
246 *    Examples: fitsBits(5,3) = 0, fitsBits(-4,3) = 1
247 *    Legal ops: ! ~ & ^ | + << >>
248 *    Max ops: 15
249 *    Rating: 2
250 */
251 int fitsBits(int x, int n) {
252     int num = 33 + ~n;
253     int result = x << num;
254     result = result >> num;
255     result = result ^ x;
256
257     return !result;
258 }
259
~/datalab-handout/bits.c [utf-8,unix] [+] [c] 1,243/288 89%
-- INSERT --
```

- 처음에 shift할 횟수를 결정한다. result를 만들어 shift를 하고 다시 원래대로 되돌려서 원형과 같은지 XOR을 이용해 확인을 한다. 0이 나오면 $x == result$ 이고 1이 나오면 $x != result$ 이다. 그러므로 !result를 반환한다.

8. isLessOrEqual

```
bits.c + (~/datalab-handout) - VIM
263 /*
264 * isLessOrEqual - if x <= y then return 1, else return 0
265 *    Example: isLessOrEqual(4,5) = 1.
266 *    Legal ops: ! ~ & ^ | + << >>
267 *    Max ops: 24
268 *    Rating: 3
269 */
270 int isLessOrEqual(int x, int y) {
271     int xx = (x >> 31) & 1;
272     int yy = (y >> 31) & 1;
273     int yx = y + (~x + 1);
274     int check = (yx >> 31) & 1;
275     int result = (xx & !yy) | (!(xx ^ yy) & !check);
276
277     return result;
278 }
279 /*
~/datalab-handout/bits.c [utf-8,unix] [+] [c] 16,277/293 94%
-- INSERT --
```

- x와 y의 부호를 얻은 후 yx에 $y-x$ 를 계산하고, yx의 부호 비트를 얻는다. $x \leq y$ 가 성립할 조건은 (1) $x < 0, y > 0$, (2) 두 값은 모두 부호 비트가 같고 check는 양수를 나타낼 때. 이므로 그 값을 반환한다.

9. rotateLeft

```
bits.c (~/datalab-handout) - VIM
306 /*
307 rotateLeft - Rotate x to the left by n
308 *   Can assume that 0 <= n <= 31
309 *   Examples: rotateLeft(0x87654321,4) = 0x76543218
310 *   Legal ops: ~ & ^ | + << >> !
311 *   Max ops: 25
312 *   Rating: 3
313 */
314 int rotateLeft(int x, int n) {
315     int a, b, c;
316
317     a = 32 + ~n;
318     b = ~(~0 << n);
319
320     c = (x >> a) >> 1;
321     c = b & c;
322
323     x = x << n;
324
325     return x | c;
326 }
327 // #include "ilog2.c"
~/datalab-handout/bits.c [utf-8,unix][c] 2,307/330 99%
```

- (a = 32 + ~n) : 32 -n의 수를 찾는다.
- (b) : 왼쪽 비트가 1이 되게 하여 not를 취해 오른쪽 n비트만큼 1이 되게 한다.
- © : c를 비트로 이동시킨다.
- (x = x << n) : x를 회전된 비트로 만든다.
- x | y를 하여 반환한다.

10. 결과화면

```
a201502023@2018-sp: ~/datalab-handout
gcc -O -Wall -m32 -lm -o btest bits.c btest.c decl.c tests.c
btest.c: In function 'main':
btest.c:528:9: warning: variable 'errors' set but not used [-Wunused-but-set-variable]
    int errors;
        ^
5. Running './dlc -e' to get operator count of each function.

Correctness Results      Perf Results
Points  Rating  Errors  Points  Ops      Puzzle
1       1       0       2       4       bitAnd
2       2       0       2       3       getByte
3       3       0       2       6       logicalShift
4       4       0       2       39      bitCount
1       1       0       2       1       isZero
2       2       0       2       2       isEqual
2       2       0       2       6       fitsBits
3       3       0       2       16      isLessOrEqual
3       3       0       2       10      rotateLeft

Score = 39/39 [21/21 Corr + 18/18 Perf] (87 total operators)
a201502023@2018-sp:~/datalab-handout$
```