

2016 시스템 프로그래밍
- Datalab -

제출일자	2016.10.03
분 반	00
이 름	정윤수
학 번	201302482

결과 1 ./dlc bits.c

./dlc bits.c 한 결과.

```
a201302482@localhost: ~/datalab-handout
a201302482@localhost:~/datalab-handout$ ./dlc bits.c
/usr/include/stdc-predef.h:1: Warning: Non-includable file <command-line> included from includable file /usr/include/stdc-predef.h.

Compilation Successful (1 warning)
a201302482@localhost:~/datalab-handout$
```

결과 2 ./btest

./btest 한 결과.

```
a201302482@localhost: ~/datalab-handout
a201302482@localhost:~/datalab-handout$ ./btest
Score  Rating  Errors  Function
1       1       0      bitAnd
2       2       0      getByte
3       3       0      logicalShift
4       4       0      bitCount
1       1       0      isZero
2       2       0      isEqual
2       2       0      fitsBits
3       3       0      isLessOrEqual
3       3       0      rotateLeft
Total points: 21/21
a201302482@localhost:~/datalab-handout$
```

결과 3 ./driver.pl(./driver.pl -u 학번)

./driver.pl 한 결과.

```
a201302482@localhost: ~/datalab-handout

Compilation Successful (1 warning)

4. Compiling and running './btest -g -r 2' to determine performance score.
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.

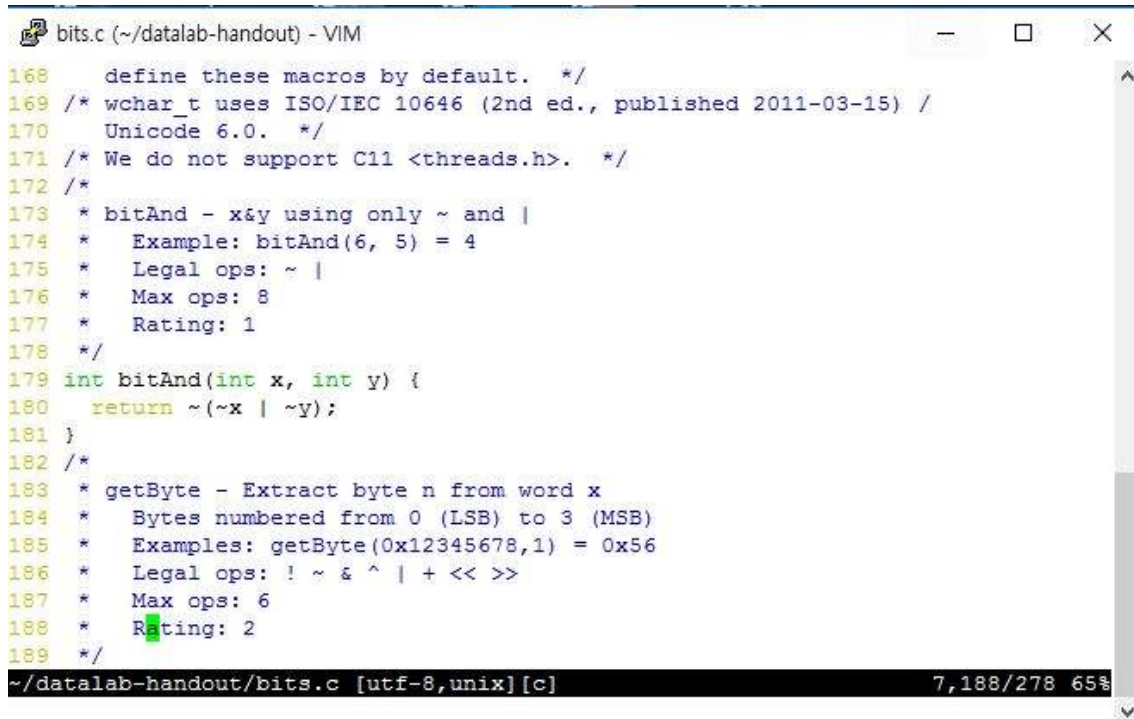
Correctness Results      Perf Results
Points Rating Errors Points Ops      Puzzle
1      1      0      2      4      bitAnd
2      2      0      2      3      getByte
3      3      0      2      6      logicalShift
4      4      0      2      31     bitCount
1      1      0      2      2      isZero
2      2      0      2      2      isEqual
2      2      0      2      8      fitsBits
3      3      0      2      18     isLessOrEqual
3      3      0      2      11     rotateLeft

Score = 39/39 [21/21 Corr + 18/18 Perf] (85 total operators)
a201302482@localhost:~/datalab-handout$
```

소스코드 설명

1 bitAnd(int x, int y)

소스코드 스크린샷



```
bits.c (~/datalab-handout) - VIM
168  define these macros by default.  */
169  /* wchar_t uses ISO/IEC 10646 (2nd ed., published 2011-03-15) /
170  Unicode 6.0.  */
171  /* We do not support C11 <threads.h>.  */
172  /*
173  * bitAnd - x&y using only ~ and |
174  *   Example: bitAnd(6, 5) = 4
175  *   Legal ops: ~ |
176  *   Max ops: 8
177  *   Rating: 1
178  */
179  int bitAnd(int x, int y) {
180      return ~(~x | ~y);
181  }
182  /*
183  * getByte - Extract byte n from word x
184  *   Bytes numbered from 0 (LSB) to 3 (MSB)
185  *   Examples: getByte(0x12345678,1) = 0x56
186  *   Legal ops: ! ~ & ^ | + << >>
187  *   Max ops: 6
188  *   Rating: 2
189  */
~/datalab-handout/bits.c [utf-8,unix][c] 7,188/278 65%
```

이 함수는 x와 y의 And연산을 OR와 NOT연산만으로 하는 프로그램이다. x와 y의 And를 |와 ~으로만 만들려면 논리회로에서 배운 법칙을 사용해야 할거라고 생각을 하였다. OR의 NOT은 AND가 됨으로 괄호 밖에서 전체적으로 NOT을 해주면 AND를 만들 수 있다.

```
bits.c (~/datalab-handout) - VIM
185 * Examples: getByte(0x12345678,1) = 0x56
186 * Legal ops: ! ~ & ^ | + << >>
187 * Max ops: 6
188 * Rating: 2
189 */
190 int getByte(int x, int n) {
191     return x >> (n << 3) & 255;
192 }
193 /*
194 * logicalShift - shift x to the right by n, using a logical shift
195 * Can assume that 0 <= n <= 31
196 * Examples: logicalShift(0x87654321,4) = 0x08765432
197 * Legal ops: ! ~ & ^ | + << >>
198 * Max ops: 20
199 * Rating: 3
200 */
201 int logicalShift(int x, int n) {
202     return (x >> n) & ~(((1 << 31) >> n) << 1);
203 }
204 */
205 /*
206 * bitCount - returns count of number of 1's in word
~/datalab-handout/bits.c [utf-8,unix][c] 7,206/278 71%
```

이 함수는 x의 값의 n번째 바이트의 값을 출력을 해주는 함수이다. 1바이트는 8비트임으로 8자리를 차지한다. 내가 원하는 바이트를 얻으려면 원하는 바이트를 뒤에서부터 8개의 비트가 되게 Shift를 시키고 바이트에서 모든 수가 1인 255와 AND연산을 하여 그 바이트의 값을 알아낸다 0번째 바이트가 필요하다면 Shift연산을 할 필요가 없고 1번째 바이트가 필요하다면 오른쪽으로 8개 만큼 Shift연산을 하게 한 후 255와 AND를 하게 되면 그 바이트의 값을 알 수 있다.

소스코드 설명

3 logicalShift(int x, int n)

```
bits.c (~/datalab-handout) - VIM
197 * Legal ops: ! ~ & ^ | + << >>
198 * Max ops: 20
199 * Rating: 3
200 */
201 int logicalShift(int x, int n) {
202     return (x >> n) & ~(((1 << 31) >> n) << 1);
203 }
204 }
205 /*
206 * bitCount - returns count of number of 1's in word
207 * Examples: bitCount(5) = 2, bitCount(7) = 3
208 * Legal ops: ! ~ & ^ | + << >>
209 * Max ops: 40
210 * Rating: 4
211 */
212 int bitCount(int x) {
213     int sum = 0;
214     sum = sum + (x & 15);
215     sum = sum + ((x >> 4) & 15);
216     sum = sum + ((x >> 8) & 15);
217     sum = sum + ((x >> 12) & 15);
218     sum = sum + ((x >> 16) & 15);
~/datalab-handout/bits.c [utf-8,unix][c] 7,218/278 76%
```

음수가 Right Shift를 하면 logical적으로는 0이 차지하지만 산술적으로는 1이 들어가게 된다. 그러므로 이 함수는 1이 들어가게 되는 것을 0으로 바꿔야만 한다. 1을 왼쪽으로 31번 Shift연산을 하면 가장 작은수가 된다. 그것을 이용하여 x가 쉬프트 한 것 만큼 같이 쉬프트 연산을 해준다. 그리고 NOT연산을 이용하면 SHift연산을 한 만큼의 자리가 0으로 차게 되는데 그것을 x와 AND연산을 하게되면 logical SHIFT한 값을 얻을 수 있게 된다.

```
bits.c (~/datalab-handout) - VIM
212 int bitCount(int x) {
213     int result;
214     int a = 0x1;
215
216     a = a | (a<<8);
217     a = a | (a<<16);
218
219     result = a & x;
220     result += a & (x>>1);
221     result += a & (x>>2);
222     result += a & (x>>3);
223     result += a & (x>>4);
224     result += a & (x>>5);
225     result += a & (x>>6);
226     result += a & (x>>7);
227
228     result += result >> 16;
229     result += result >> 8;
230
231     result = result & 0xFF;
232     return result;
233 }
```

~/datalab-handout/bits.c [utf-8,unix][c] 3,212/288 79%

이 함수는 매개변수로 받은 x 의 비트에 1의 개수가 몇 개 있는지 확인을 하여 개수를 반환을 해주는 함수이다. 개수를 저장할 변수 `result`와 개수를 구하기 위해서 `a`를 선언을 한다. `a`의 1번째 비트, 9번째 비트, 17번째 비트 25번째비트를 1로 만들어 주고 x 와 AND연산을 하면서 1일 있는지 없는지 확인을 하며 7비트를 RIGHT SHIFT 한다. 그 후 RIGHT SHIFT연산을 이용하여 32~17비트의 값을 16~1비트로 옮겨 주고 16~9비트의 값을 8~1비트로 옮겨준다. 그러면 비트가 1인 것의 총합의 개수는 8~1비트 안에 들어있으므로 0xFF와 AND연산을 하여 그 값을 반환을 해준다.

소스코드 설명

5	isZero(int x)
---	---------------

```
bits.c (~/datalab-handout) - VIM
227 * Legal ops: ! ~ & ^ | + << >>
228 * Max ops: 2
229 * Rating: 1
230 */
231 int isZero(int x) {
232     return !(x ^ 0);
233 }
234 /*
235 * isEqual - return 1 if x == y, and 0 otherwise
236 * Examples: isEqual(5,5) = 1, isEqual(4,5) = 0
237 * Legal ops: ! ~ & ^ | + << >>
238 * Max ops: 5
239 * Rating: 2
240 */
241 int isEqual(int x, int y) {
242     return !(x ^ y);
243 }
244 /*
245 * fitsBits - return 1 if x can be represented as an
246 * n-bit, two's complement integer.
247 * 1 <= n <= 32
248 * Examples: fitsBits(5,3) = 0, fitsBits(-4,3) = 1
~/datalab-handout/bits.c [utf-8,unix][c] 7,248/278 88%
```

x의 값이 0인지 아닌지 구별을 하는 함수이다. Exclusive OR 연산을 사용하여 x와 0이 같은지 같지 않은지 확인을 할 수 있다. x가 0이라면 모든 값이 0임으로 !에 의해서 1이 반환이 됨으로 참이된다. 만약 x의 값이 0이 아니라 다른 수라면 0이 아닌 다른 수가 될것임으로 !연산에 의해서 0이 반환이 될 것이다.


```

bits.c (~/datalab-handout) - VIM
239 *   Rating: 2
240 */
241 int isEqual(int x, int y) {
242     return !(x ^ y);
243 }
244 /*
245 * fitsBits - return 1 if x can be represented as an
246 * n-bit, two's complement integer.
247 *   1 <= n <= 32
248 *   Examples: fitsBits(5,3) = 0, fitsBits(-4,3) = 1
249 *   Legal ops: ! ~ & ^ | + << >>
250 *   Max ops: 15
251 *   Rating: 2
252 */
253 int fitsBits(int x, int n) {
254     return !((x << (33~n) >> (33+~n))^x) ;
255 }
256 /*
257 * isLessOrEqual - if x <= y then return 1, else return 0
258 *   Example: isLessOrEqual(4,5) = 1.
259 *   Legal ops: ! & ^ | + << >>
260 *   Max ops: 24
~/datalab-handout/bits.c [utf-8,unix][c] 2,260/278 92%

```

isEqual 함수 또한 위의 isZero와 같은 방식으로 Exclusive OR 연산과 !연산을 이용하여 값을 구한다. x와 y의 값이 만약 일치한다면 모든 비트의 값들이 0이 될 것이다. 그러므로 !연산으로 인해 1을 반환을 하게 될 것이다. 두 개의 값이 서로 다르다면 0이 아닌 수가 bit안에 들어가 있음으로 !연산에 의해 0이 반환이 될 것이다.

소스코드 설명

7 fitBits(int x, int n)

```
bits.c (~/datalab-handout) - VIM
251 * Rating: 2
252 */
253 int fitsBits(int x, int n) {
254     return !((x << (33+~n) >> (33+~n))^x) ;
255 }
256 /*
257  * isLessOrEqual - if x <= y then return 1, else return 0
258  * Example: isLessOrEqual(4,5) = 1.
259  * Legal ops: ! & ^ | + << >>
260  * Max ops: 24
261  * Rating: 3
262  */
263 int isLessOrEqual(int x, int y) {
264     int sy = (y>>31) & 1;
265     int sx = (x>>31) & 1;
266     return ((!(sy^sx)) & ((x+~y) >>31 & 1)) | ((!sy) & sx);
267 }
268 /*
269  * rotateLeft - Rotate x to the left by n
270  * Can assume that 0 <= n <= 31
271  * Examples: rotateLeft(0x87654321,4) = 0x76543218
272  * Legal ops: ~ & ^ | + << >> !
~/datalab-handout/bits.c [utf-8,unix][c] 7,272/278 97%
```

이 함수는 x의 값이 n비트로 x의 보수 값을 표현을 할수있는지 없는지 판별해서 값을 반환을 해주는 함수이다. 비트는 총 32자리이다 하지만 n비트로 표현을 할수있는지 없는지 알기 위해서는 나머지 비트 만큼 밀고 다시 당겨 와서 밀리고 당겨진 수가 원래의 값과 같으면 그 값은 n비트의 값으로 표현을 할 수 있는 수이다. 32비트의 -n만큼을 왼쪽으로 Shift하고 오른쪽으로 Shift 해야한다 32 + n은 비트로 32 + ~n +1임으로 33+~n만큼의 비트를 이동시키며 비트를 이동시킨 값과 원래 값이 같음 유무를 Exclusive연산을 이용하여 확인을 하고 값이 같으면 비트의 값이 0임으로 !연산을 이용하여 1을 반환을 해준다.

소스코드 설명

8 isLessOrEqual(int x, int y)

```
bits.c (~/datalab-handout) - VIM
267      * isLessOrEqual - if x <= y then return 1, else return 0
268      *   Example: isLessOrEqual(4,5) = 1.
269      *   Legal ops: ! & ^ | + << >>
270      *   Max ops: 24
271      *   Rating: 3
272      */
273 int isLessOrEqual(int x, int y) {
274     int sy = (y>>31) & 1;
275     int sx = (x>>31) & 1;
276     return (((!(sy^sx)) & ((x+~y+1) >>31 & 1)) | ((!sy) & sx)) | !(x^y);
277 }
278 /*
279 * rotateLeft - Rotate x to the left by n
280 *   Can assume that 0 <= n <= 31
281 *   Examples: rotateLeft(0x87654321,4) = 0x76543218
282 *   Legal ops: ~ & ^ | + << >> !
283 *   Max ops: 25
284 *   Rating: 3
285 */
286 int rotateLeft(int x, int n) {
287     return (x << n) | ((x >> (32+(~n +1))) & ~((~1+1)<< n));
288 }
~/datalab-handout/bits.c [utf-8,unix][c] 3,287/288 Bot
```

이 함수는 $x \leq y$ 를 판별을 하여 값이 참이면 1을 반환해 주고 값이 참이 아니면 0을 반환을 해주는 함수이다. 먼저 y의 값이 양수고 x의 값이 음수이면 무조건 1을 반환하게 한다. $(!sy) \& (sx)$ 를 이용하면 이것을 구현할수 있고, x, y 의 부호의 값이 서로 같다면 y의 2의 보수연산을 한 것의 x의 값을 더하여 그 값의 부호를 확인을 한다. 그 부호의 값이 1이면 음수임으로 y의 값이 더 큰 것 이고 값이 0이면 x의 값이 더 큰것이임으로 0을 반환을 해준다. x와 y가 같을때를 고려 하여 $!(x^y)$ 를 OR연산으로 고려를 해준다..

소스코드 설명

8 rotateLeft(int x,int y)

```
bits.c + (~/datalab-handout) - VIM
265     int sx = (x>>31) & 1;
266     return ((!(sy^sx)) & ((x~y) >>31 & 1)) | (!(sy) & sx);
267 }
268 /*
269  * rotateLeft - Rotate x to the left by n
270  *   Can assume that 0 <= n <= 31
271  *   Examples: rotateLeft(0x87654321,4) = 0x76543218
272  *   Legal ops: ~ & ^ | + << >> !
273  *   Max ops: 25
274  *   Rating: 3
275  */
276 int rotateLeft(int x, int n) {
277     return (x << n) | ((x >> (32+(~n+1))) & ~((~1+1)<< n));
278 }
279
280
281
282
283
284
285
286
~/datalab-handout/bits.c [utf-8,unix][+][c] 1,286/286 Bot
-- INSERT --
```

이 함수는 x의 값이 Left Shift연산을 사용하면 왼쪽 끝의 있던 값이 오른쪽 끝으로 가게 만드는 함수이다. 이렇게 할려면 왼쪽에서 Shift되어 없어지게 될 값들을 저장을 한 후 OR연산으로 x에 다시 값을 넣어 준다. -1을 LeftShift하면 오른쪽 끝의 값은 0이 된다 그것의 NOT연산을 하면 오른쪽 끝의 값은 1이되게 하고 나머지는 0이 되게 만들 수 있다. x의 LeftShift연산으로 없어지게 될 값들을 같은 만큼의 LeftShift연산을 한 -1의 NOT연산으로 값을 저장을 하고 OR연산으로 x의 값을 다시 넣어주면 오른쪽 끝에 없어진 값들을 넣을수 있다.