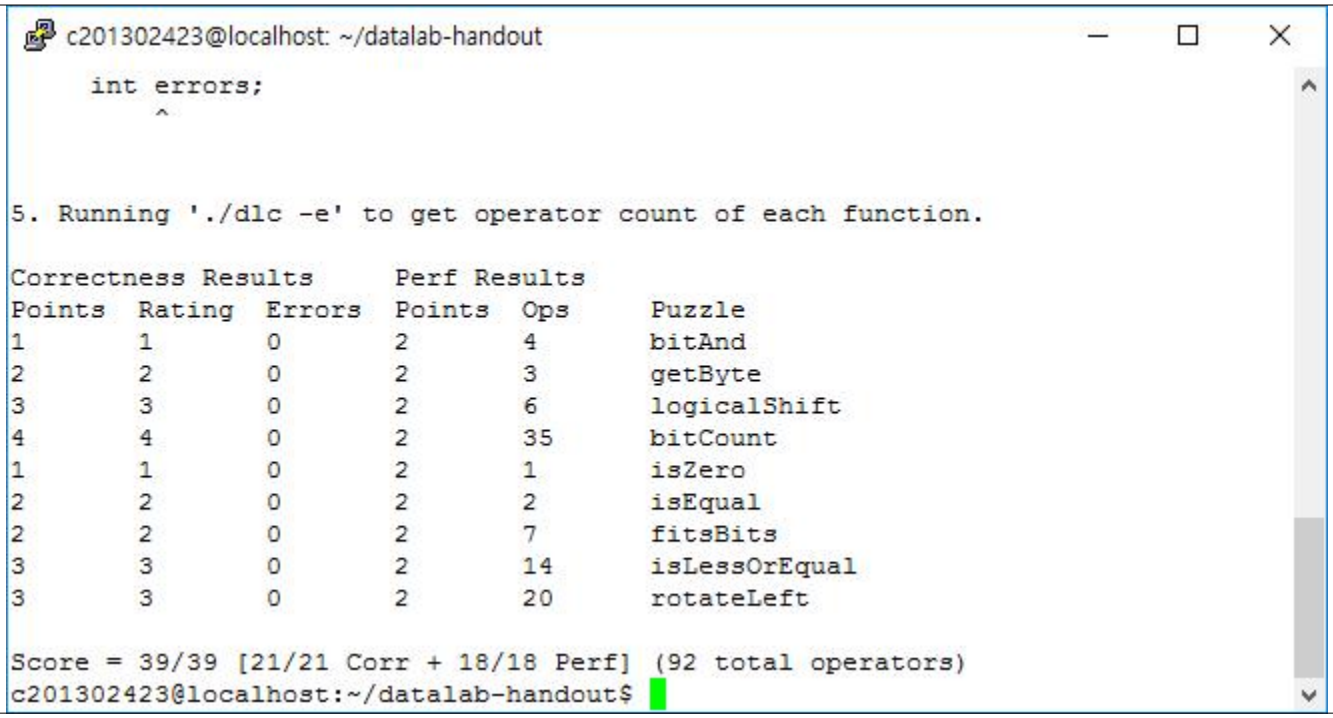
2016년 시스템 프로그래밍

- HW 02 -

제출일자	2016.10.04.
이　　름	신종욱
학　　번	201302423
분　　반	02



1. bitAnd : not과 or을 이용해서 and 만들기

```
int bitAnd(int x, int y) {
        return ~(~x|~y);
}
```

드모르간의 법칙을 이용해서 쉽게 만들 수 있다.

2. getByte : 정수에서 n번째 바이트 리턴하는 함수만들기.

```
int getByte(int x, int n) {
        int byte = 255&(x>>(n<<3));
        return byte;
}
```

비트 연산자는 비트 단위에서 계산하기 때문에 n*8을 해서 정수를 옮겨야하는데하는데 곱하기 연산자는 불허하기 때문에 같은 의미인 <<3을 n에게 해서 만든다음 그 바이트를 제외한 다른 숫자들은 필요없기 때문에 8비트가 1로 꽉찬 255와 and해주면 그 외 바이트들은 0이되고 필요한 바이트를 추출할 수 있다.

3. logicalShift : shift를 이용해서 산술 rightshift가 아닌 논리적 rihgtshift를 구현하기

```
int logicalShift(int x, int n) {
        int min = 1<<31;
        return(x>>n)&~((min>>n)<<1);
}
```

여기서 x를 shift 시킬 때 주의할점은 바로 부호비트이다 부호비트가 1일 경우 rihgtshift를 할 경우 산술 shift(int 형일 경우)하기 때문에 0채워지는게 아니라 1이 채워진다.
그래서 논리적으로 바꾸긴위해서 앞의 1들을 다 0으로 바꿔줘야한다.
그러기 위해선 옮기는 n-1만큼 0을 앞에 채우고 그뒤는 1로채우는 비트를 만들어서 and를 해주면된다.
그래서 min을 1로 선언후 31칸 Leftshift해주면 정수형의 최솟값이된다
오른쪽으로 다시 n만큼 옮기면 1로채우면서 shift를 하면 한자리 더이동하기 때문에 다시 왼쪽으로 1칸 이동후 1의보수를 취하면 0이 n-1개만큼 채우고 나머지는 1로 채우는 비트가 돼서 and를 해주면된다.

4. 정수를 2진수로 나타냈을 때 1을 개수를 세는 bitcount를 구현하기

```
int bitCount(int x) {
int a = 1;
int b = 1<<8;
int c = 1<<16;
int d = 1<<24;
int hak = a+b+c+d;
int count = (x&hak)+((x>>1)&hak)+((x>>2)&hak)+((x>>3)&hak)+((x>>4)&hak)
   +((x>>5)&hak)+((x>>6)&hak)+((x>>7)&hak);
        int upupupbit = count>>24;
        int upupbit = count>>16;
        int upbit = count>>8;
        int result = (upupupbit+upupbit+upbit+count)&255;
return result;
}
```

처음에는 x를 한칸씩 이동하면서 계속 비교해서 더했었다.
하지만 그럼 연산자 개수가 많아져서 성능이 떨어졌다. 그렇다고 bit를 4개로 쪼겠더니 저장할 수 있는 최대숫자가 15라서 1의 개수가 많아지면 오류가 났다. 8bit가 가장 적절하여서 채택하였다.
0000 0001 0000 0001 0000 0001 0000 0001 이렇게 비트를 만든후 x를 한칸씩 옮기면서 비교해서 더하였다.
그럼 각각 8비트 단위로 1의 개수가 나오는데 더할땐 8칸씩 이동하면서 더해야 우리가 원하는 10진수로 개수 더하기가 된다. 그리고 난뒤 8비트 위에 숫자는 무시해서 더해야 하기 때문에 and를 해주었다.

5. 정수 x가 0인지 아닌지 판단하여서 맞다면 1를 아니면 0을 리턴하는 메소드

```
int isZero(int x) {
        return !x;
}
```

일단 1또는 0을 반환하는 메소드를 만들라고해서 0이면 1을 리턴하고 다른숫자라면 0을 나타내는 !연산자를 써서 이용해야 한다고 생각했다.
그러므로 !x를 리턴하면 x가 0이라면 1을 리턴 그 외라면 0을 리턴한다.

6. x,y와 같다면 1 다르다면 0을 리턴하는 메소드

```
int isEqual(int x, int y) {
int a = x^y;
        return !a;
}
```

이번에도 무조건 !연산자는 필수이다.
x,y가 같다면 0이 되도록해야 쉽게 코드를 완성할 수 있다 거기에 알맞은 연산자는 xor이다.
xor은 다르면 비트에 1을 남기고 같으면 0을 남기는 비트연산자이다.
xy를 xor하면 비트가 모두 같다면 0으로 채워질꺼고 한 개라도 다르면 1이 남아서 0이 아닌 숫자가 될 것이다.

7.x를 2의보수로 바꿀 때 필요한 비트수가 n개 이하인지 판단하여 가능하면 1 불가능이면 0을 리턴하는 메소드이다.

```
int fitsBits(int x, int n) {
int a=(x>>(n+(~0)));
return !a|!(a+1);
}
```

일단 n비트가 있으면 그비트로 나타낼 수 있는 최대값+1=최솟값의 절대값이 같다는걸 아는게 중요하다 그러므로 만약 정수 x가 양수이면 x가 구성하는 비트수+1(음수 표현용) 만큼 있으면 가능하다.
음수에서는 이미 음수를 표현, 즉 msb가 1이고 나머지가 0으로 채워져있으면 2의 보수를 채우는게 불가능하다.
그러니 둘의 경우를 따로 해서 구하면된다.
x가 양수의 경우에는
여기서 x를 n-1만큼 이동시켜야한다 왜냐하면 n비트중 한비트는 부호비트로 사용되기 때문이다.
n-1을 만드는 방법중에 하나가 n+(~0)로 표현하면 된다. 그러므로 !a를 써서 양수의 가능성을 판단하고
x를 n-1만큼 오른쪽으로 이동시켜서 만약 부호비트외 모두 0이 된다면
x는 (n-1)개로 2의보수를 표현하고 한 개의 부호비트(1)가 있는 값이라는 것을 뜻한다.
음수의 경우에는
반대로 n-1만큼 오른쪽으로 이동시켜서 모두 1이되면 표현가능하다.
모든비트가 1이라는 말은 즉 –1를 나타내기 때문에 !(a+1)를 써서 음수의 가능성을 판다하였다.
양음수의 가능성을 or하면 원하는 리턴값이 나온다.

8.정수 y가 x보다 크거나 같으면 1반환 아니라면 0반환

```
int isLessOrEqual(int x, int y) {
        int a = x^y;
          int b = x & (~y);
          int c = (~(x ^ y)) & (x + (~y + 1));
          return (!a)|(((b | c) >> 31) & 1);}
```

같을 때 1반환은 앞에서 사용해서 equal 메소드를 똑같이 해주면된다.
작다를 판단해야한다. 그건 a를 통해서 나타내었고
음수 – 양수일경우에도 항상 1을 반환해야 하기 때문에 b에서 x&~y를 하면 부호비트가 1로 될꺼기 때문에
31right shift하면 1이남아 반환된다.
그다음은 둘이 부호가 같을 때 즉 양수-양수나 음수-음수일 때 서로 계산하면 만약 y가 더클 경우 그값의 부호
비트는 1이 될 것이다.
그걸 이용해서 만든 코드가 c이다.
~(x ^ y)로 인해 부호비트가 같으면 부호비트는 1을 반환하고 아닐경우는 0반환한다.
x+(~y+1)는 x-y를 뜻함으로 y가크다면 부호비트가 1일 것이다. 이둘을 AND하면
부호가 같고 x<y인 경우 부호비트가 1로 유지된다.
각각의 경우를 or해서 retrun하면 된다.
음수 – 양수는 생각 할필요없음으로 코드에 넣지않아도된다.

9.x를 LeftShift할 경우 사라지지않고 한바퀴 돌아 다시 앞으로 오는 순환Shift를 만들어라

```
int rotateLeft(int x, int n) {
int a = 0x7f;
          return
(((x>>1)&((a<<24)+(((a<<1)+1)<<16)+(((a<<1)+1)<<8)+((a<<1)+1)))>>(31+(~n+1)))+(x<<n);
}
```

x를 n만큼 왼쪽으로 옮기면 n만큼 왼쪽 비트가 사라지고 오른쪽에 n개의 0이 생기기 시스템이기 때문에
사라지기전에 그비트들을 저장한뒤 더해주면된다.
여기서 a는 0111 1111로 부호비트는 0이고 나머지는 1인 비트를 만들어서 x>>1한 것관 and해주면 x의 0번째
비트빼고는 다 저장되었다(n은 31이하이기 때문에 0번째비트는 저장 안해도된다)
이미 한칸을 이동했음으로 32-n이 아닌 31-n만큼 이동시키면 순환해서 돌아오는 비트들이 된다.
31-n임은 31+(~n+1)로 나타낼 수 있다.
이저장된 비트는 31-n만큼 오른쪽으로 이동시키고 x를 n만큼 이동시킨것과 더하면 된다.