Section 11

리틀엔디안과 빅엔디안

dumpcode() 함수에 대해 알게 된 구타는 대견하게도
실습을 하기로 마음 먹습니다.
구타는 먼저 4바이트짜리 정수형 변수를 하나 만든 후,
그것을 memory dump를 해보기로 했습니다.

```
#include “dumpcode.h”

int main()
{
        int test = 10;

        dumpcode(
```

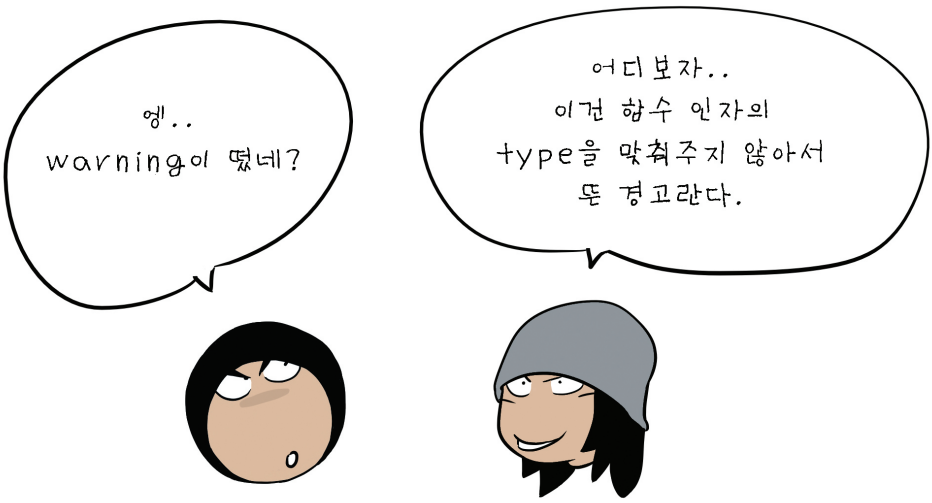
소스코드를 완성한 구타는 컴파일을 시도합니다.

./11/ex1.c

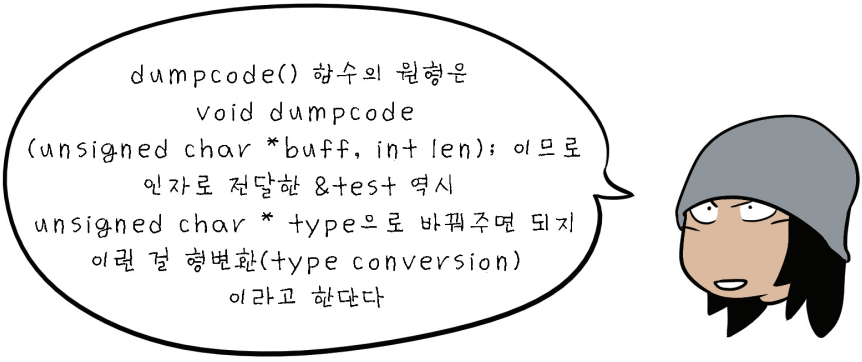
```
#include “dumpcode.h”

int main()
{
        int test = 10;

        dumpcode(&test, 4);
}
```

[컴파일 결과]

```
$ cd 11
$ gcc -o ex1 ex1.c
ex1.c: In function `main':
ex1.c:7: warning: passing arg 1 of `dumpcode' from incompatible pointer type
$
```



11.리틀엔디안과 빅엔디안



./11/ex2.c

```
#include “dumpcode.h”

int main()
{
        int test = 10;

        dumpcode((unsigned char *)&test, 4);
}
```

[컴파일 결과]

```
$ gcc -o ex2 ex2.c
$
```

dumpcode()는 잘 작동했지만, 구타는 무언가 이상한 점을 발견합니다.

```
$ ./ex2
0xbffffb44  0a 00 00 00                                  ....
$
```

11.리틀엔디안과 빅엔디안

이처럼 메모리에 저장된 값을 dump해보면, 그 순서가 반대로 나타나는 재미있는 특징을 볼 수 있습니다.
이를 컴퓨터 용어로 “Little Endian”이라고 부르는데요, 이 중 Endian이라는 단어는 영어사전을 찾아봐도 나오질 않습니다.

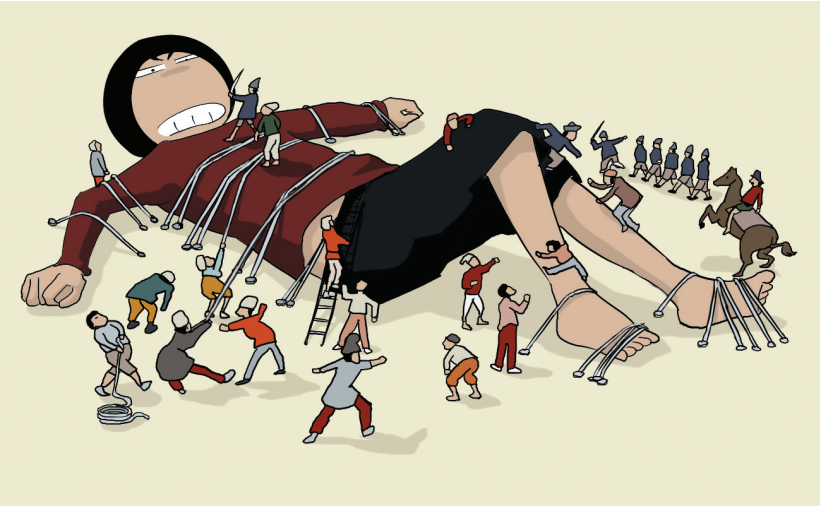
그 이유는 이 “Little Endian”이라는 말이 “걸리버 여행기”라는 다들 잘 알고 계실 소설에서 처음 만들어졌기 때문인데요, 그 이야기가 재미있기 때문에 소개를 해드리려고 합니다.

“걸리버 여행기”란, 항해술과 의술을 공부한 주인공이 선박의 선원들을 돌보는 외과의사로 취업한 후, 항해 도중 겪게 되는 황당한 사건들에 대한 이야기를 담고 있는 모험 소설입니다.

이 소설은 1726년 조나단 스위프트란 영국 작가에 의해 쓰여졌는데요, 가상의 이야기에 빗대어 인간의 추악한 모습들을 비유적으로 풍자하며 오늘날까지도 세계적인 명작에 꼽히고 있습니다.

소설의 주인공은 항해 중 사고로 인해 “소인국 나라”, “거인국 나라”, “하늘의 섬” 등에 표류하게 되는데요, “Little Endian”, 그리고 그에 대립되는 “Big Endian” 이라는 용어는 이들 중 “소인국 나라”편에 등장합니다.

거대한 폭풍우에 휩쓸려 난파된 선박에서 탈출한 주인공은, 끝내 외딴 섬에 도착해 정신을 잃게 됩니다. 그리고 다음날 정신을 차렸을 땐 이미 온몸이 밧줄로 꽁꽁 묶인 상태였습니다.



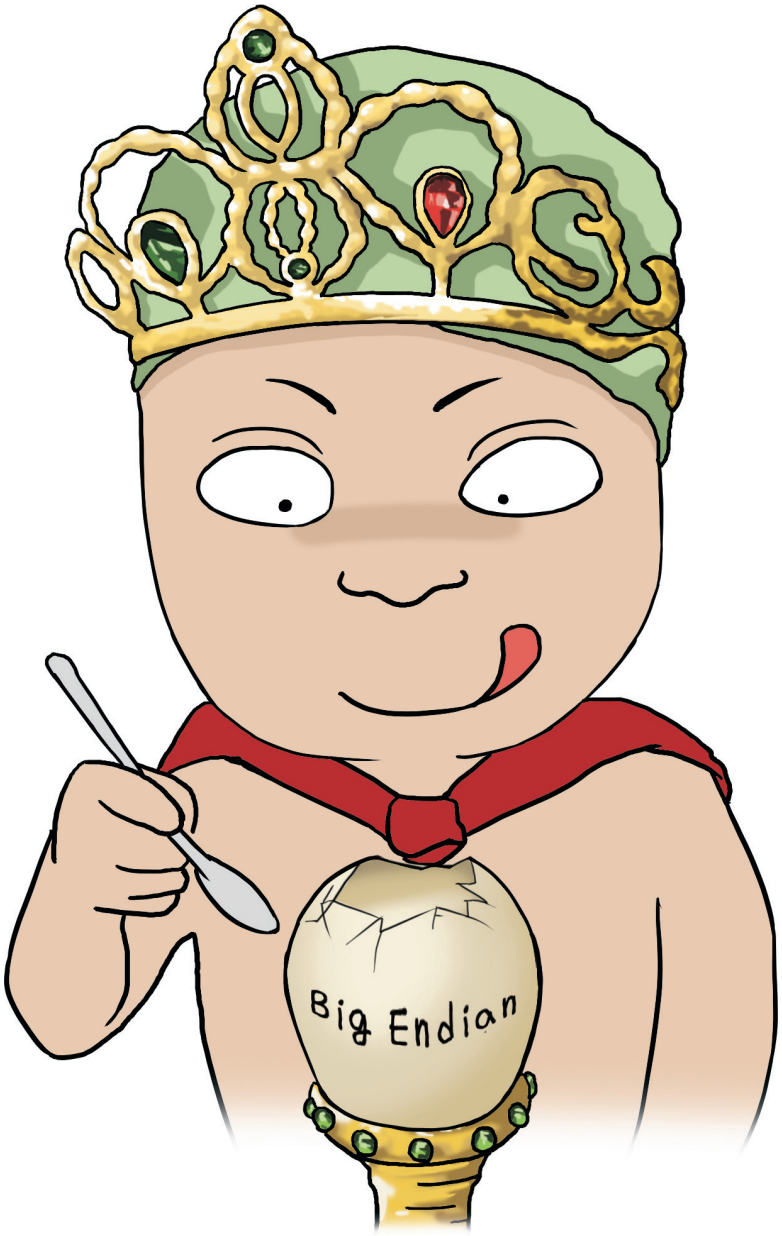
주인공은 밧줄에서 벗어나려고 발버둥을 쳐보지만, 수 많은 소인국 군인들의 화살 공격에 기가 죽어 순순히 복종을 하게 됩니다. 이후 주인공은 발에 쇠사슬이 묶인 채 노예와 같은 생활을 하게 되는데요, 친절함과 너그러움으로 점차 소인국 사람들로부터 신뢰를 얻어 자유의 몸이 된 후, 이웃 국가와의 전쟁을 승리로 이끌기도 하며, 급기야 두 국가를 화해 모드로 이끌고, 결국엔 그 섬을 빠져나와 고향으로 돌아간다는 훈훈한 이야기를 담고 있습니다.

"Little Endian"과 "Big Endian" 이야기는 위에 언급 된 두 국가인 릴리퍼트와 블레프수크가 도대체 무엇 때문에 사이가 안 좋아졌는지에 대해 듣게되는 부분에서 등장하는데요, 그 사연은 이렇습니다.

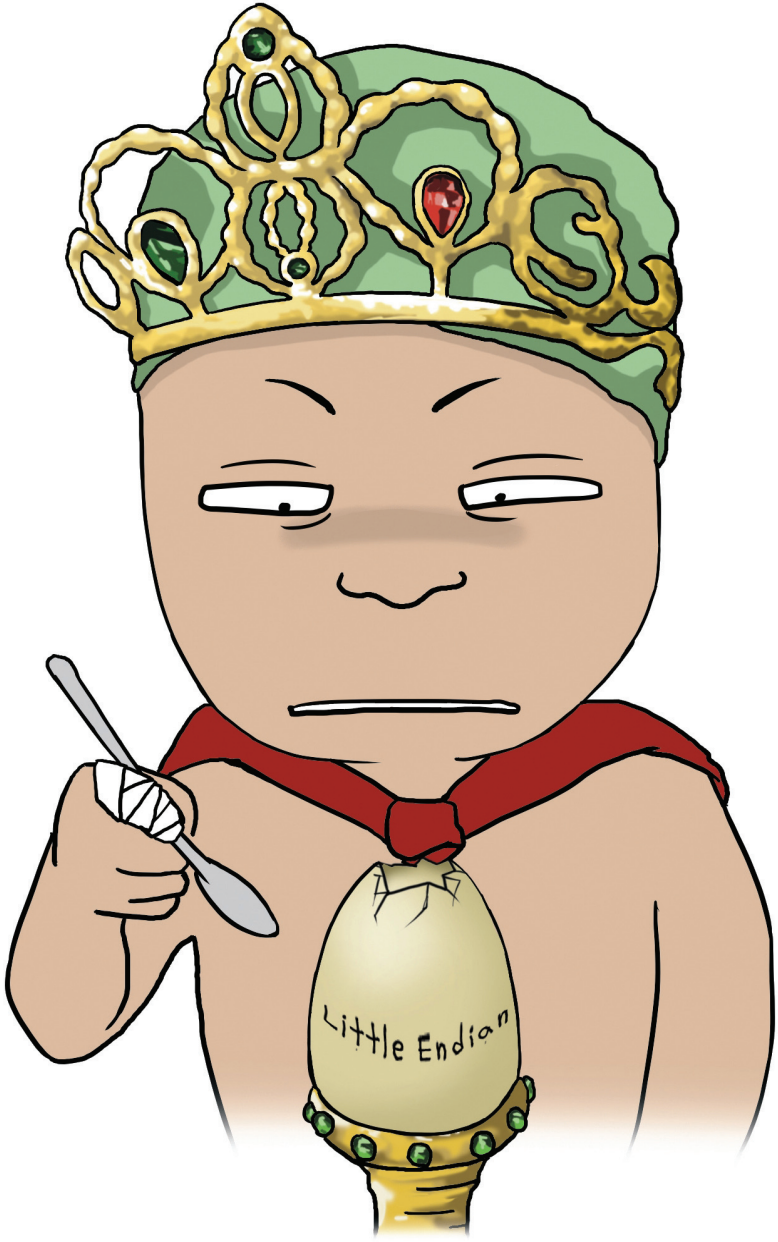
약 36개월 전, 릴리퍼트 제국은 지금과 마찬가지로 식사 때 달걀을 즐겨 먹었습니다.
그리고 이들은 달걀을 먹을 때, 끝 부분이 넓직한 부분을 위로 향하게 한 다음 껍질을 깨먹었다고 합니다.

그런데 어느날 역사적인 사건이 발생하는데, 바로 황제의 세자가 이 넓직한 부분부터 달걀을 깨어먹다가 그만 껍질에 손을 베어버린 것입니다.
화가 난 황제는 앞으로 달걀을 먹을 때 넓직한 부분이 아닌, 뾰족한 부분부터 깨어먹어라, 그렇지 않는자는 사형에 처한다라는 새로운 법률을 공포하게 됩니다.

11.리틀엔디안과 빅엔디안



11.리틀엔디안과 빅엔디안



11.리틀엔디안과 빅엔디안

하지만 이에 반대하는 무리가 생겨나기 시작했고, 여러 번의 반란 끝에 국가는 두개의 당파로 갈라지게 됩니다.

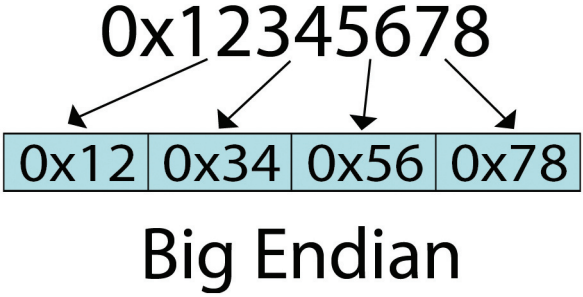
그리고 뾰족한 끝 부분, 즉 작은 부분에서부터 깨어 먹는 것을 주장하는 무리를 "Little Endian", 반대로 넓직한 끝 부분에서부터 깨어 먹어야 한다고 주장하는 이들은 "Big Endian"이라고 부르게 된 것입니다.

"Big Endian" 당파는 이웃 국가인 블레프수크의 도움을 받거나 망명을 가면서 급기야 두 국가간의 전쟁으로 발전합니다.

이처럼 소인국 국가들은 작은 몸짓답게 참으로 사소한 것을 가지고 싸움을 하고 있는데요,이는 마찬가지로 사소한 것들로 전쟁을 하고 살인을 하는 우리 인간들을 풍자한 것이라고 합니다. (당시 영국과 프랑스의 종교 전쟁을 빗댄 것이라는 해석이 있습니다.)

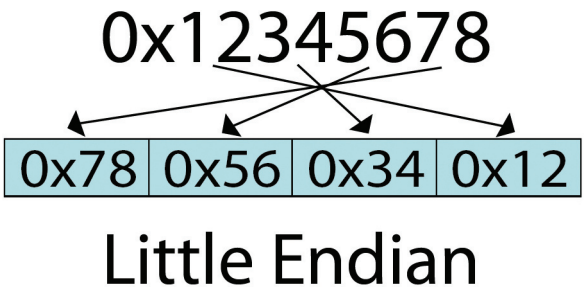
이제 다시 컴퓨터 분야로 넘어옵시다.
이 "Little Endian"과 "Big Endian"이란 용어는 Danny Cohen이라는 사람의 논문 중 Memory Order라는 섹션에서 처음 인용됩니다.
이 논문에선 CPU가 메모리에 데이터를 저장할 때 어느 순서로 저장하는가에 대해 설명하는데, 이 때 왼쪽에서 오른쪽 순서로 저장하는 것을 "Big Endian", 그리고 오른쪽에서 왼쪽으로 저장하는 것을 "Little Endian"이라고 비유한 것이 시초가 되어 지금까지도 사용되어지고 있는 것입니다.

이처럼 메모리에 값을 저장하는 두 방법엔 서로 장단점이 있는데요,
먼저 높은 쪽의 값을 먼저 저장하는 Big Endian을 봅시다.



Big Endian은 두 개의 숫자를 비교할 때 더욱 빠르게 연산할 수 있습니다.
만약 첫 바이트에서 한쪽이 더 크다면 나머지 바이트들은 더 이상 비교를 할 필요가 없기 때문입니다. 그리고 Big Endian은 우리가 사용하는 방식에 친근하다는 장점도 있습니다.

다음으로 Little Endian입니다.
Little Endian은 이 수가 짝수인지 홀수인지를 검사할 때 빠르다는 장점이 있습니다.



첫 바이트 하나만 보면 짝수인지 홀수인지를 금방 알 수 있기 때문입니다. 그리고 연산 과정에서 자릿수 증가가 생겼을 때 Big Endian 방식보다 더 빠르다고 합니다.

그리고 또 하나, 이 Little Endian 방식은 포인터(pointer)의 값 참조 시 유리합니다.
예를 들어, char *x라는 포인터 변수가 하나 있고, 이 x가 가리키는 메모리 주소의 값이 0x12345678라고 가정해 봅시다. 이 때, y = *x와 같이 포인터 변수가 가리키는 값을 참조하고자 할 경우, 포인터의 시작 주소에서 1바이트(char형)만 가져오면 자연스럽게 가장 낮은 바이트인 0x78가 참조되게 됩니다. 왜냐면 Little Endian의 특성에 따라 메모리에 저장된 데이터 값은 0x78 0x56 0x34 0x12가 되기 때문입니다.

이와 같은 바이트 저장 순서를 Byte Ordering이라고 부르는데요,
만약 서로 다른 Byte Ordering을 사용하는 두 PC가 네트워크 통신을 하면서 데이터를 주고 받는다면 이 Byte Ordering을 통일화 시켜주는 작업이 필요할 것입니다.

11.리틀엔디안과 빅엔디안

현재 TCP/IP 표준은 Big Endian이며, 그래서 대부분의 프로그래밍 언어가 Byte Ordering을 Big Endian 타입으로 바꿔주는 htons()와 htonl() 함수를 제공하고 있기도 합니다.

CPU 개발 업체에 따라 Big Endian과 Little Endian이 구분되는데요, 다음과 같습니다.

Little Endian	Big Endian
Intel x86	IBM
AMD	SPARC
DEC	Motorola

Little Endian과 Bin Endian 사이엔 서로만의 장점이 있기도 하고,
기술적인 관점을 떠나 생각해봐도 어느 것이 더 좋다고 선뜻 말 할 수는 없습니다.

예를 들어 한글이나 영어는 왼쪽에서 오른쪽으로 글자를 쓰지만,
히브리어, 아랍어, 이집트 상형 문자 등은 오른쪽에서 왼쪽으로 글자를 씁니다.
이를 가지고 누가 옳다라고 말 할 수 없는 것과 같기 때문입니다.

여튼 메모리 분석을 정확하게 하기위해선 ,
이 Big Endian과 Little Endian을 잘 알고 있어야 합니다.