System Programming
숙제 1

인천대학교 | 컴퓨터공학부

시스템프로그래밍 | 지도교수 박문주

송병준 | 201701562

0.목차

System Programming 숙제 1

CSAPP 2장 문제 풀이

송병준
2019년 9월 16일

1. 과제

Computer System, A Programmer's Perspective 2장 연습 문제 풀이
- 2.60
- 2.64
- 2.67

2. 환경

운영체제: macOS 10.14.6 (Darwin Kernel Version 18.7.0 x86_64)
프로세서: Intel Core i5
편집기: vim 8.0.1365
빌드: Apple LLVM version 10.0.1 (clang-1001.0.46.4)

3. 문제 풀이

3.1. 문제 2.60

w비트 워드를 0(가장 덜 중요한 바이트)부터 $w/8 - 1$(가장 중요한 바이트)까지 번호를 매긴다고 하자. 비부호형 값의 i번째 바이트를 b로 교체한 값을 리턴하는 C 함수를 다음과 같이 작성하시오.

```
unsigned replace_byte (unsigned x, int i, unsigned char b);
```

다음은 이 함수가 어떻게 동작해야 하는지를 보여주는 예제들이다.

```
replace_byte(0x12345678, 2, 0xAB) --> 0x12AB5678
replace_byte(0x12345678, 0, 0xAB) --> 0x123456AB
```

비부호형 정수 안의 특정 바이트를 바꾸는 함수이다.
바이트 단위 접근을 위해 인자 x의 주소를 unsigned char 형의 포인터로 해석하도록 한다. 그렇게 얻은 포인터는 메모리 안에서 정수 x 중 가장 낮은 메모리 주소에 위치한 바이트를 가리키게 된다. 해당 주소에 오프셋 i를 더하면 교체할 바이트위 위치를 확보할 수 있다.
이때 시스템이 Little-Endian인지 또는 Big-Endian인지 여부에 따라 접근할 바이트의 순서가 달라질 수 있다. 예를 들어 PC에 통상적으로 사용되는 x86 또는 x64 시스템에서는 인자 i로 0을 넘겨 해당 함수를 실행했을 때에 LSB를 포함하는 바이트가 바뀐 채로 반환된다.

구현은 아래와 같다.

```
/**
 * 2.60
 * Change a byte at specific index i in x, with b.
 *
 * @param x      the source to be changed.
 * @param i      index of byte to change in x.
 * @param b      the replacement.
 *
 * @return the result after the replacement,
 * or the original x if the i is out of range.
 */
unsigned replace_byte(unsigned x, int i, unsigned char b) {
      if (i < 0 || i > sizeof(unsigned) - 1) {
            // Conversion from int to size_t is safe.
            // i will always be positive when evaluating
            // the expression at the right.
            return x;
      }

      unsigned char *target = (unsigned char *)&x + (unsigned)i;
      *target = b;

      return x;
}
```

실행 결과는 다음과 같다.

```
==== Test 0 Started ====
Assertion succeeded: replace_byte(0x12345678, 2, 0xAB) == 0x12AB5678
Assertion succeeded: replace_byte(0x12345678, 0, 0xAB) == 0x123456AB
==== Test 0 Succeeded ====
```

3.2. 문제 2.64

다음 함수를 구현하는 프로그램을 작성하시오.

```
/* Return 1 when any odd bit of x equals 1; 0 otherwise.
   Assume w=32 */
int any_odd_one(unsigned x);
```

함수는 자료형 int가 $w = 32$비트라고 가정하는 것을 제외하고는 비트수준 정수 코딩 규칙(164쪽)을 따라야 한다.

홀수 비트가 하나라도 1이면 1을, 그렇지 않으면 0을 반환하는 함수이다.
어떤 비트가 홀수 비트인지 짝수 비트인지 판단하기 위해서 해당 비트의 위치를 참고해야 한다. 비트의 위치는 LSB를 1로 하여 LSB부터 센다. 예를 들어 0b1010에서 짝수 비트는 모두 1로, 홀수 비트는 모두 0으로 설정되어 있다.
홀수 비트를 검출하기 위해서는 검출 대상과 같은 폭을 가진 홀수 비트 마스크를 사용하면 된다.

한 바이트 안에서 홀수 비트만 1인 경우 0b0101, 즉 0x55로 표현된다. 이를 32비트로 확장하면 비트 마스크는 0x55555555이다. 마스크를 적용한 후 not 연산을 두 번 취하여 0 또는 1을 얻어낸다.

구현은 다음과 같다.

```
/**
 * 2.64
 * Check if the given integer has any odd bits.
 * Assume that integer width is 32.
 *
 * @param x      the target to determine.
 *
 * @return       non-zero if any odd bits, or zero.
 */
int any_odd_one(unsigned x) {
      return !(!(x & 0x55555555));
}
```

실행 결과는 다음과 같다.

```
==== Test 1 Started ====
Assertion succeeded: any_odd_one(0x00050000) == 1
Assertion succeeded: any_odd_one(0xAAAAAAAA) == 0
==== Test 1 Succeeded ====
```

3.2. 문제 2.67

int가 32비트인 머신에서 실행하면 1을 출력하고, 아니면 0을 출력하는 함수 int_size_is_32()를 작성하려고 한다. sizeof연산자는 사용할 수 없다. 다음은 대략적으로 작성한 것이다:

```
1   /* The following code does not run properly on some machines */
2   int bad_int_size_is_32() {
3       /* Set most significant bit (msb) of 32-bit machine */
4       int set_msb = 1 << 31;
5       /* Shift past msb of 32-bit word */
6       int beyond_msb = 1 << 32;
7
8       /* set_msb is nonzero when word size >= 32
9          beyond_msb is zero when word size <= 32  */
10      return set_msb && !beyond_msb;
11  }
```

32비트 SUN SPARC에서 컴파일하고 실행하면 이 프로시저는 0을 리턴한다. 다음과 같은 컴파일러 메시지가 이 문제에 대한 설명을 해주고 있다:

```
warning: left shift count >= width of type
```

A. 이 코드가 왜 C 표준을 지키지 못하였는가?

B. 코드를 수정해서 자료형 int가 최소 32비트인 모든 머신에서 잘 동작하도록 하라.

C. 코드를 수정해서 자료형 int가 최소 16비트인 모든 머신에서 잘 동작하도록 하라.

sizeof 연산자 없이 현재 시스템의 int형 크기를 알아내는 함수이다.
MSB까지 비트를 shift하여 비트가 범위 밖으로 나갔는지 검출함으로써 정수형의 크기를 알 수 있다. 예를 들어 32비트 시스템의 경우 LSB로부터 31번째 비트에 1을 설정하면 이는 non-zero의 값을 가지게 된다. 이때 LSB로부터 32번째 비트에 1을 설정하면 이는 32비트 표현 범위의 밖이므로(부호 불문) zero의 값을 가지게 된다.
그러나 C언어에서 정수형은 해당 형의 크기만큼 shift 연산을 취할 수 없다. 다시 말해 32비트 int에 〈〈 32 연산은 취할 수 없다. 하지만 그보다 작은 양만큼 여러 번 shift하는 것은 가능하다. 1이라는 부호형 정수 상수에 좌 shift를 32번 취하고 싶다면 먼저 16번 shift한 뒤 다시 16번 shift할 수 있다. C의 shift 연산자는 순서대로 적용되기 때문에 variable 〈〈 32와 variable 〈〈 16 〈〈 16은 동치이다.
만약 for문을 사용한다면 정수가 zero가 되는 시점까지 1씩 shift하는 방법으로 그 크기를 알아낼 수 있다. for문을 사용하지 않는다면 가장 작은 정수 크기로 예상되는 값보다 하나 작은 수 만큼씩 여러번 shift를 취하면 된다(int가 16비트 시스템이라면 15씩).

구현은 아래와 같다.

```c
/**
 * 2.67
 * Check if this machine has 32 bits of int size.
 *
 * Implementation limit: No use of sizeof.
 *
 * @return non-zero if size of int is 32, or zero.
 */
int int_size_is_32() {
      int set_msb = 1 << 15 << 15 << 1;
      int beyond_msb = set_msb << 1;

      return set_msb && !beyond_msb;
}
```

실행 결과는 다음과 같다.

```
==== Test 2 Started ====
Assertion succeeded: int_size_is_32() == 1
==== Test 2 Succeeded ====
```

4. 마무리

연습문제는 [컴퓨터 시스템 3판]에서 발췌했습니다.

일부 솔루션은 다음 저장소를 참고했습니다:
https://github.com/DreamAndDead/CSAPP-3e-Solutions

이 보고서와 문제 풀이는 다음 저장소에 업로드되어 있습니다:
https://github.com/potados99/CSAPP3e-solutions