REPORT

보고서 작성 서약서

1. 나는 타학생의 보고서를 복사(Copy)하지 않았습니다.
2. 나는 타학생의 보고서를 인터넷에서 다운로드 하여 대체하지 않았습니다.
3. 나는 타인에게 보고서 제출 전에 보고서를 보여주지 않았습니다.
4. 보고서 제출 기한을 준수하였습니다.

나는 보고서 작성시 위법 행위를 하지 않고,
성.균.인으로서 나의 명예를 지킬 것을 약속합니다.

과　　목 : 프로그래밍 기초와 실습
과 제 명 : 실습 4 : Functions
담당교수 : 민 형 복 교수
학　　과 : 자연과학계열
학　　년 : 1학년
학　　번 : 2014310407
이　　름 : 이준혁
제 출 일 : 2014년 10월 18일

1, Introduction

이번 실습에서는 Function, 즉 함수에 대해 공부한다. 이 때 이 함수라는 것은 수학에서 말하는 그 함수와 비슷하다. 수학에서의 함수는 정의역의 값에 하나의 치역의 원소가 대응되는 것인데, C언어에서의 함수도 마찬가지로 입력을 받아서, 출력을 내놓는 형태이다. 또한, 수학에서의 함수는 정의역의 원소를 식에 대입해 치역의 원소에 대응시키는데, C언어에서의 함수 역시 입력을 가공하여 출력을 내놓는다. 이렇게 입력 받은 값을 어떻게 가공해서 출력으로 내놓는 지가 함수의 내용이 되겠다. 다만, C언어의 함수는 입력이 없거나, 출력이 없거나, 혹은 둘 다 없을 수도 있다는 점에서 수학에서의 함수와 약간 다르다.

이런 함수를 사용하는 이유는 무엇일까? 그것은 top – down design 때문이다. Al Kelly, Ira Pohl이 집필한 C by Dissection 4th edition, 123쪽에 따르면, "Top – down design consist of repeatedly decomposing a problem into smaller problems"이라 되어 있다. 즉, 큰 문제를 작은 문제로 나누는 것이 top – down design이고, 이렇게 작은 문제로 나누어서 문제를 풀 때 필요한 도구가 바로 함수이다. 이런 이유 때문에 함수를 사용하는 방법을 익혀야 하고, 그것이 이 실습의 이유이다.

함수는 function prototype, function call, function definition인 3단계로 이루어져 있는데, function prototype은 main 함수 위에, function definition은 main 함수 밑에 온다. (function prototype을 쓰지 않고 function definition이 main 함수 위에

올 수도 있긴 하지만 이 경우는 style이 나쁘므로 항상 main 함수 아래에 온다고 하자.) 이전까지의 실습에서는 헤더파일 이외의 모든 프로그램을 main함수 안에서만 작성했지만, function은 그렇지 않다는 점을 주의해서 프로그램을 작성해야 한다.

2. Problem Statement

(A) Describe what the problem is

factorial (n!)과 exponent (a^n) 라는 두 개의 함수를 만들어 실행시켜야 한다. 이에 대한 상세한 설명은 다음과 같다.

1. factorial (n!)

   어떤 정수 하나를 입력 받아서, 그 수 보다 작은 모든 1 이상의 정수를 곱한 값을 출력하는 함수이다.

   즉, 이해하기 쉽게 표현하면 n! = 1 * 2 * … * n 이다. 다만, 이는 1이상의 정수에만 해당되는 내용이고, n=0일 때 0! 은 1으로 둔다. n이 0보다 작다면 이 방법으로 계산할 수 없으므로 이 때는 정의하지 않는다.

   for loop 를 이용해 이 함수를 작성해야 한다.

2. exponent (a^n)

   정수로 된 밑과 지수를 입력 받아서 밑을 지수 번 곱해서 나온 값을

출력하는 함수이다.

즉, 이해하기 쉽게 표현하면 a^n = a * a * … * a (단, a는 n개) 이다. 이 방식대로 하면 n이 음수일 때는 계산할 수 없으므로 제외한다. 그리고 a^0은 a를 1에다 0번 곱했다는 뜻이므로 1이 된다. 다만 0^0은 이렇게 정의하기에 문제가 있으므로 역시 정의하지 않는다. 즉, 지수가 0미만 이거나 지수와 밑이 동시에 0인 경우는 배제한다.

while loop 를 이용해 이 함수를 작성해야 한다.

이렇게 정의된 두 함수를 이용해 사용자에게 구하고 싶은 factorial 값과 exponent값을 입력하도록 해야 한다. 그런데 위에서 언급한, 정의되지 않는 경우로 입력할 때는 오류 메시지를 주고 재입력을 요구해야 한다. 단, 사용자는 정수만 입력한다고 가정한다.

그리고 또 factorial과 exponent의 입력을 따로 받아서, 각 입력마다 입력의 오류가 생기면 다시 입력하도록 요구해야 한다. 즉, factorial 입력을 받고, 오류가 있으면 exponent 를 입력 받기 전에 오류 메시지를 주고 재입력을 요구해야 한다.

(B) Describe how you solve the problem.

가장 먼저 문제의 핵심이 되는 두 개의 함수 factorial과 exponent함수를 어떻게 만들 지 생각해 보았다. for 문을 이용해 n보다 작은 모든 정수를 곱하는 것과

while 문을 이용해 a를 n번 곱하는 것은 이미 저번 실습 flow of control에서 다뤘던 것이므로 별로 어렵지 않을 것이라 생각했다. 그래서 이 부분은 걱정하지 않았다.

다만 잘못된 입력을 줬을 때 어떻게 오류 메시지를 줄까에 대해 고민했다. 우선 첫 번째 입력(factorial)을 받고, 오류가 있으면 두 번째 입력(exponent)을 받기 전에 오류 메시지를 줘야 했으므로 factorial과 exponent 두 개의 section 으로 나누어서 생각하기로 했다. 즉, factorial의 입력이 문제가 없으면 exponent 로 넘어가도록 말이다.

factorial에서는 n<0인 경우만 배제하면 됐으므로, while문을 이용하면 좋을 것이라 판단했다.

그러나 exponent에서는 while문을 이용하기 좀 그랬다. 오류가 일어나는 경우가 지수가 음수일 때, 밑과 지수가 모두 0일 때 두 개가 있어서, 이 두 문장을 하나의 while문으로 쓰기에는 너무 복잡해 질 것 같았고, 순서대로 두 개의 while문을 쓰기에는 두 번째 검사하는 부분에서 오류가 나 재입력 할 때에 첫 번째 오류가 나게 입력을 한다면 낭패였기 때문에, 이 두 방법 다 별로인 것 같았다. 따라서, 두 경우를 if문으로 만들어서 하나의 while문에 넣는 게 가장 나을 것이라 생각했다.

3. Implementation

가장 먼저 function prototype에서 함수의 이름을 설정하는 작업을 해야 했는데, 이는 문제에서 제시한대로 각각 factorial과 exponent로 입력했다.

그 다음에 각각의 함수에 사용될 변수의 이름을 설정해야 했다. factorial이라는 함수에 들어갈 두 개의 변수, 즉 n에 해당하는 입력 변수와 n!에 해당하는 출력 변수 두 개가 필요했다. 이 이름은 각각 factorial안에 들어가는 수라는 이름으로 factorial_num, factorial을 해서 나온 결과라는 뜻으로 factorial_result라고 명명했다. 이 부분은 민형복 교수님의 template p4.c 를 참고했다.

exponent라는 함수에 들어가는 변수의 이름 역시 같은 방법으로 설정했다. 다만 여기에서는 a^n에 해당하는 입력이 a, n으로 두 개라는 점을 유념했다. 따라서 a에 해당하는 변수는 '밑'에 들어가는 수 이므로 base_num, n에 해당하는 변수는 '지수' 에 들어가는 수이므로 exponent_num으로 작성했다. 그리고 a^n은 exponent를 해서 나온 결과라는 의미로 exponent_result라고 두었다. 이 부분 역시 민형복 교수님의 template p4.c 를 참고했다.

사용자가 입력하는 값과, 그 입력을 통해 출력되는 값이 모두 정수이기 때문에, 모두 정수형 변수로 선언했다.

그 다음에 문제 해결 방법을 생각했을 때처럼 두 개의 section으로 나누었다. 첫 번째 section은 factorial이었다. 우선 사용자에게 factorial_num 값을 입력을

요구하고, 받아서 올바른 입력인지 검사하기로 했다. while문을 이용해 조건식에 factorial_num < 0으로 두고, while문 내부를 재입력을 요구하도록 했다. 이렇게 하면 factorial_num이 0 보다 작으면 항상 재입력이 나올 것이기 때문이었다. 이 과정을 끝내면 factorial_num값이 항상 0 이상이 될 테고, factorial_result = factorial(factorial_num); 이라는 function call을 이용해 factorial_result를 출력하기로 했다. 이 때 factorial이라는 함수는 추후에 작성하기로 했다. 만약 함수가 올바르다면, 이 과정을 통해 얻어지는 factorial_result는 항상 옳은 값을 가진다고 할 수 있다. 그 다음 이 값을 바로 출력하지 않고 두 번째 section의 내용인 exponent로 넘어갔다.

exponent 역시 factorial에서 했던 것과 마찬가지로, 사용자에게 base_num과 exponent_num 값의 입력을 요구했고, 그 후 올바른 입력인지 검사하기로 했다. 다만 여기서는 검사할 때 문제 해결 방법에서 언급한 대로 while문과 if문을 이용하기로 했다. 다음과 같이 작성했다.

```
while (1) {
    if (exponent_num<0) {
        지수가 0보다 작을 수 없다는 오류 메시지 출력 후 재입력 요구
    }
    else if ((exponent_num == 0) && (base_num == 0)){
        0의 0승은 정의되지 않는다는 오류 메시지 출력 후 재입력 요구
    }
    else {
```

```
        break;
    }
}
```

이 방법대로 하면, 지수가 0보다 작거나, 지수와 밑이 동시에 0일 때는 while문을 탈출하지 못하므로, 두 경우가 아닐 때까지 재입력을 요구하리라 생각했다. 즉, 만약 정상적인 입력이 되면 break로 loop를 빠져 나올 수 있다고 보았다. 사실 break문을 이용하지 않고도 while문의 조건식을 변경 함으로서 정상적인 입력일 때 while문을 나갈 수도 있었겠지만, 그 방법들은 너무 읽고 이해하기가 힘들어서, 깔끔하게 break문을 이용했다. 이 과정을 끝내면 항상 올바른 입력이 될 테고, 여기서도 exponent_result = exponent(base_num, exponent_num); 이라는 function call을 이용해 exponent_result를 출력하기로 했다. factorial과 함께 exponent라는 함수 역시 function definition에서 작성하기로 했다. 이 역시 함수가 올바르다면, exponent_result는 문제에서 제시한 대로 옳은 값을 가진다고 할 수 있다.

이 과정이 끝나면 모든 변수들이 올바르게 값이 할당되었으므로, 문제에서 요구한 대로 값들을 출력하고 main함수를 끝내기로 했다. 그리고 그 아래에 function definition을 작성했다.

우선 factorial의 function definition을 다음과 같이 작성했다. factorial이라는 함수는 입력으로 정수 n을 받아들여서 n!을 출력하는 함수이다. 따라서, 일단

정수 factorial_num(편의상 n으로 두자)을 입력으로 받아 들였다. 그리고 for문을 통해 순차적으로 곱을 표현하기 위한 변수로 숫자를 의미하는 number라는 정수형 변수를 선언했다. 이 numebr가 1부터 n까지 커지면서 곱을 하기 위함이다. 그리고 factorial의 결과를 계산하고 반환하기 위해 위에서 사용한 변수와 같은 이름의 factorial_result라는 변수를 선언하면서 1로 초기화했다. 즉, 다음과 같다.

```
int number, factorial_result = 1;
```

이 때 factorial_result라는 main함수에서의 변수와 이름이 같은 변수를 사용했는데, 이는 main함수에서의 factorial_result라는 변수와 function definition에서의 factorial_result라는 변수가 다르기 때문에 가능했다.

그리고 for 문을 이용해 number라는 변수가 1부터 n이 될 때까지 1을 더하면서 factorial result에 곱하려 했다. 이렇게 하면 1보다 크거나 같은 정수 n에 대하여 for문의 끝에서 factorial result의 값이 1 * 2 * … * n이 될 것이라 생각했다. 즉, factorial result 값이 문제에서 요구한대로 n!이 나오면 내 생각이 맞는 것이다. 이 함수는 n을 받아들여서 n!을 반환하는 함수이므로 factorial result를 반환하면 될 것이라 생각했다. 즉, 다음과 같이 프로그램을 작성했다.

```
for (number = 1; number <= factorial_num; number++)
    factorial_result *= number;
return factorial_result;
```

또한 이렇게 되면 n = 0일 때 for 문을 한번도 거치지 않고 나오므로 1이 반환되므로, 0! =1 이라는 원래의 가정과 일치한다고 볼 수 있겠다.

exponent 함수는 입력으로 정수 a,b를 받아들여 a^b를 출력하는 함수이다. 그래서 base_num과 exponent_num을 받아들였다. (편의상 a, b라 두자) 그리고 while문을 이용해 a를 b번 곱하기 위해, 이 횟수를 셀, count의 약자인 cnt라는 변수를 선언하고 0으로 초기화했다. 또, exponent의 결과를 계산하고 반환하기 위해 위에서 사용한 변수와 같은 이름의 exponent_result라는 변수를 선언했고, 초기값을 1로 선언했다. 즉, 다음과 같다.

```
int exponent_result = 1, cnt = 0;
```

exponent_result 역시 main 함수에서의 exponent_result라는 변수와 이름만 같을 뿐 다른 변수였기 때문에 가능했다.

이제 while문을 이용해 a를 b번 곱하는 함수를 만들려 했다. 그렇게 하기 위해 while문의 조건식 안에 cnt를 이용한 식을 넣고, 매 루프마다 cnt가 1씩 커지게 하면서 그 때 마다 exponent_result에 a를 곱하려고 했다. 그리고 이 과정을 b번, 즉 exponent_num번 만큼 한 다음에 while문을 빠져나가려 했다. 그렇게 while문을 빠져나갔을 때 exponent_result 값이 a^b가 되어 있을 것이라 예상했다. 만약 실제로도 그렇다면 내 생각이 맞는 것이다. 그래서 이 값을 반환하기로 했다. 즉, 다음과 같다.

```
 while (++cnt <= exponent_num)
```

```
        exponent_result *= base_num;
return exponent_result;
```

또, 이렇게 되면 b = 0일 때 while 문을 한번도 거치지 않으므로 1이 반환된다. 따라서, a^0 =1 이라는 원래의 가정과 일치한다고 볼 수 있겠다. 다만, 여기서 0^0 인 경우를 이 function의 function call이 오기 전에 배제했으므로, 그 부분도 원래의 가정에 부합한다.

이렇게 짠 프로그램을 실행해 보았다.

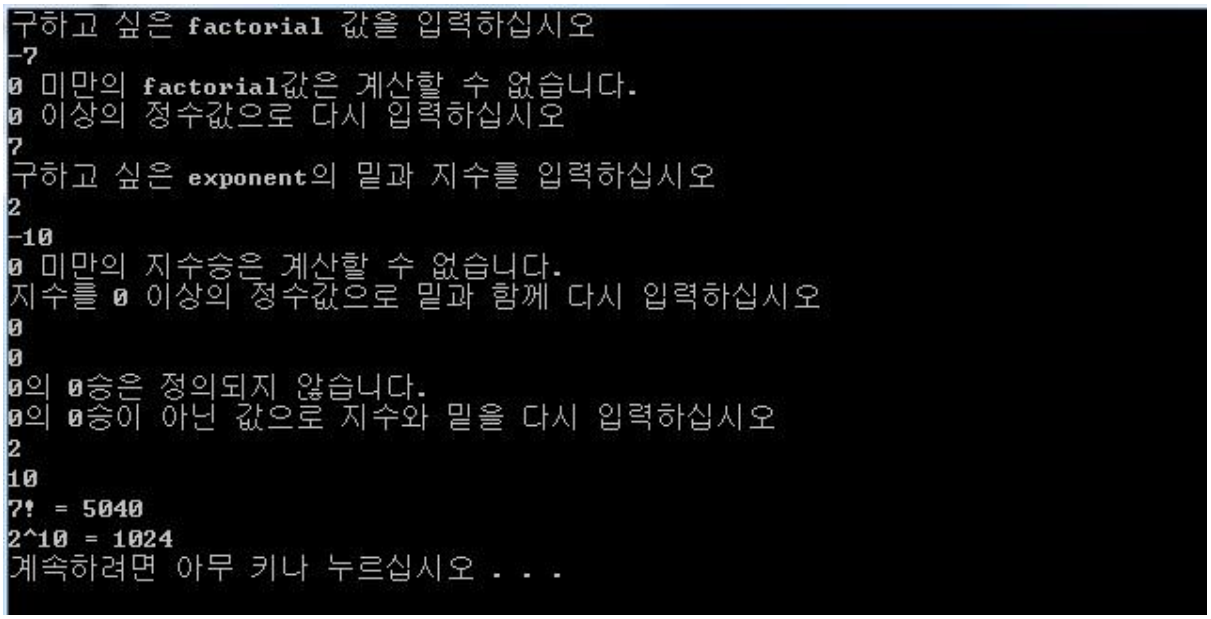
4. Result

다음 세 가지 결과가 잘 나오는지를 확인하고자 했다.

1. 올바른 값을 입력할 시 factorial과 exponent 값이 잘 출력되는가?
2. 올바르지 않은 값을 줄 시 적절한 오류 메시지를 주면서 재입력을 요구하는가?
3. 0! = 1, a^0 = 1이 제대로 출력되는가? (a는 0이 아님)

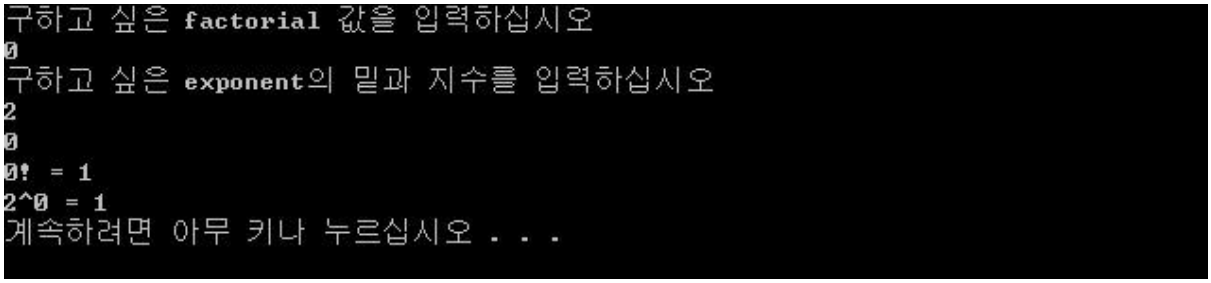
1,2번의 결과를 한번에 살펴보기 위해 factorial을 입력하는 곳에 -7, 7을 입력했고, exponent를 입력하는 곳에 (2, -10), (0, 0), (2, 10)을 입력해 보았다. 결과는 다음과

같았다.



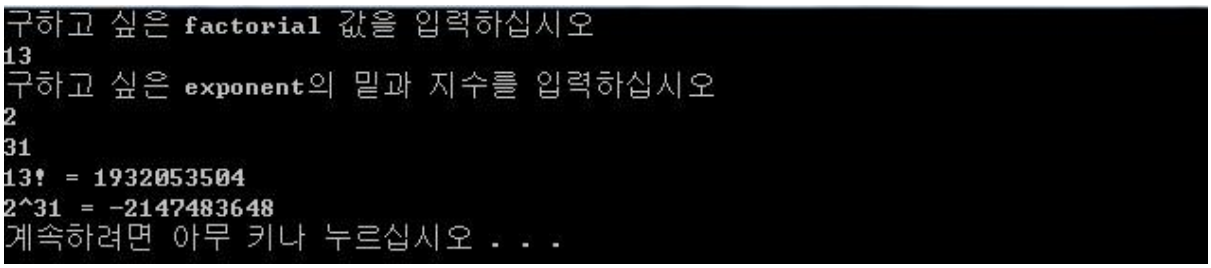
-7을 입력받을시 적절한 오류 메시지를 주면서 재입력을 요구했고, 올바른 값인 7을 주자 exponent의 입력을 요구했다. 그 후 (2, -10)을 입력하자 0미만의 지수승이 계산되지 않는다는 메시지를 출력했고, (0, 0)을 입력하자 0의 0승이 정의되지 않는다는 메시지가 출력되었다. 그 후 (2,10)을 입력하자 정상적으로 받아들였고, factorial과 exponent값이 출력되었다. 7! =5040, 2^10 = 1024라는 결과값은 실제로 계산했을 때 나오는 값과 동일하다.

한편 3번 경우, 0! =1, a^0 = 1이 제대로 출력 (a는 0이 아님)되는지 확인하기 위해 0과 (2,0)을 각각 입력해 보았다.



0! =1, 2^0 =1이라는 결과가 출력되었는데, 이는 예상한 것과 완전히 일치했다.

그러나 문제가 하나 발생했다. 바로 13! 이상의 factorial 값을 줄 때와 2^31 이상 exponent 값을 줄 때 오류가 발생했다.



원래 13! = 6227020800인데, 이 프로그램을 통해 계산한 결과로는 1932053504가 나온다. 또한 2^31 = 2147483648인데 음수인 -21474836480이 나온다.
프로그래밍한 내용만 봐서는 납득이 가지 않는 결과이다. 특히 exponent함수는 양수만 곱하는데 음수가 나오는 것은 사뭇 이상해 보였다. 어쨌든 13 이상 수를 factorial 함수에 넣거나 2^31이상의 결과값이 나오도록 수를 exponent 함수에 수들을 넣으면 원래 예상한 것과 다른 값이 나온다는 것을 확인했다.

5. Conclusion & Evaluation

이번 실습은 약간의 흠이 남았다. 결과에서도 언급했지만 13! 이상의 수와 2^31이상의 수가 결과로 나오면 결과값이 원래 예측과 다르게 나왔기 때문이다. 원래 프로그램 대로라면 분명히 13을 넣었을 때 13보다 작은 10이 13!의 약수로 들어가야 하므로, 일의 자리에 0이 오는 것이 지당한데, 이 결과는 그렇지 않다. 또, 2^31은 프로그램대로라면 2만 31번 곱하기 때문에 음수가 나올 수가 없다. 결과는 음수로 표시됐다. 게다가 모든 수에서 오류가 나는 것이 아니라 12! 이하인 수와 2^30이하인 수들에서는 정상적인 출력이 나왔기 때문에 뭐가 잘못됐는지 도통 찾을 수가 없었다.

그러나 조교님이 프로그램을 설명할 때 정수형 변수는 -2^31 ~ 2^31 내부의 수만 표현할 수 있다는 것에 착안해 생각해 보았다. 2^31 = 2147483648 이고, 13! = 6227020800 이므로, 13! > 2^31이다. 즉, 13!을 정수형 변수로 정확한 값을 표현할 수 없었을 것이고, 따라서 위와 같은 결과가 나오지 않았나 생각할 수 있겠다. 2^31 이상의 값 역시 마찬가지로 생각해 볼 수 있겠다.

이것을 깨닫고 난 후 프로그램을 어떻게 고치면 이런 점들을 제대로 잡을 수 있을까 고민해 보았다. 처음에는 결과값이 2^31보다 큰 경우를 오류 메시지를 주고 재입력하려고 했다. 그러나 조교님도 말씀하셨지만, 정수형 변수는 2^31 이상의 값을 가질 수 없으므로, 설령 그 값이 실제로도 2^31 이상의 값이어도, 출력되는 값은 다르게 나올 수 있다. 예를 들어 13!의 실제 값은 2^31 이상이지만, 출력 값은 2^31보다 작게 나온다. 따라서, 이 방법은 적절하지

못하다고 생각되었다.

이 방법 이외에도 함수를 통해 계산한 값을 검산하는 과정, 즉 while문을 이용해 factorial은 그 수 보다 작은 모든 양의 정수를 나누고, exponent는 결과를 밑으로 지수 번 나누어 나머지가 0이 아닌 경우가 있으면 오류를 출력할까 등등 여러 생각을 해 보았지만, 2^31 이상의 수가 정확히 어떤 원리에 의해 결과값이 출력되는 지 알 방도가 없어서 사용할 수가 없었다. 이 부분은 왜 이렇게 나오는지 정말 알 수가 없다. 나중에 교수님께 질문 드려 봐야겠다. 그리고 다른 방법으로 이 오류를 제거하는 방법에 대해서도 생각해 봐야겠다.

비록 위에서 언급한 오류는 있었지만, 이번 실습은 그것만 빼면 괜찮았다. 이전 실습을 마치며 함수 실습이 어렵진 않을까 걱정했지만, 괜한 걱정이었음을 이번 실습을 통해 느꼈다. 기존에 작성한 것과 달리 함수는 main function의 바깥에다 작성한다는 점과 call-by-value, 즉 function call을 할 때 값만 넘긴다는 점 정도만 조심한다면 기존에 배웠던 것들과 크게 다른 점이 없었기 때문이다.

게다가 이번 실습을 통해 얻은 것이 많다. 이 실습을 통해 function을 이용하는 법을 제대로 알았으니, 앞으로 같은 작업들을 여러 번 수행해야 할 일이 생긴다면 굳이 몇 번씩이고 같은 내용을 쓸 필요 없이 함수를 이용하면 되기 때문이다. 즉, 프로그램을 덜 지저분하고, 비교적 간단하게 짤 수 있게 되었다.

6. 참고 문헌

[1] 민형복, program template, p4.c.

[2] 프로그래밍 기초와 실습 사이트, http://class.icc.skku.ac.kr/~min/C/, 보고서 작성 요령, 2014 년 10 월.

[3] Al Kelly, Ira Pohl, *C by Dissection: The Essentials of C Programming, Fourth Edition,* Pearson, p. 123 ~ 131.

[4] 민형복, practice4.pdf.