

# Object Oriented Programming assignment1

설계 분반: 2 (신동화 교수님)

실습 분반: 1 (신동화 교수님)

학번: 2023202032

이름: 남호준

1.

## 문제 설명

1번 문제는 5개의 정수를 입력받고 다음 줄에 순서대로 최솟값, 최댓값, 평균을 출력하는 프로그램이다. 이때, 평균은 소수점 첫째 자리에서 반올림한다. (사사오입)

## 알고리즘

---

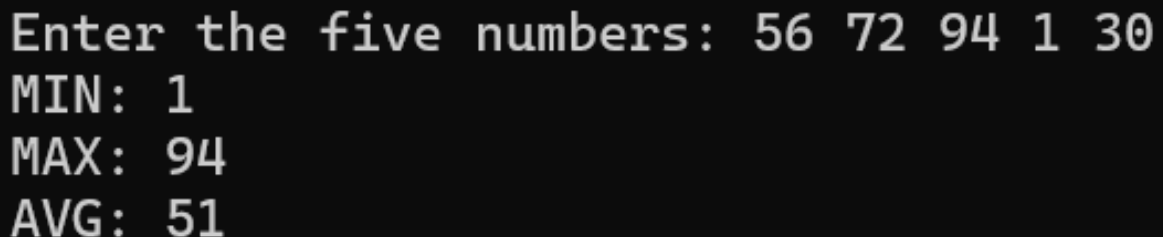
**Algorithm 1** Finding Min, Max, and Average of Five Numbers

---

```
1: Initialize variables:  $min, max, avg = 0$ 
2: Declare an array  $arr$  of size 5 and initialize all elements to 0
3: Prompt user to enter five numbers
4: for  $i \leftarrow 0$  to 4 do
5:   Read  $arr[i]$  from user input
6:   Update  $avg$  by adding  $\frac{arr[i]}{5}$ 
7:   if  $i == 0$  then
8:     Set  $min = arr[i]$  and  $max = arr[i]$ 
9:   end if
10:  if  $arr[i] < min$  then
11:    Update  $min$  to  $arr[i]$ 
12:  end if
13:  if  $arr[i] > max$  then
14:    Update  $max$  to  $arr[i]$ 
15:  end if
16: end for
17: Print " $MIN :$ "  $min$ 
18: Print " $MAX :$ "  $max$ 
19: Print " $AVG :$ "  $avg$ 
```

---

## 결과 화면



```
Enter the five numbers: 56 72 94 1 30
MIN: 1
MAX: 94
AVG: 51
```

예시와 다른 입력값을 넣어 제대로 실행이 되는지 확인하였다.

평균:  $(56+72+94+1+30)/5=253/5=50.6 \Rightarrow 51$  (소수점 첫째 자리에서 반올림)

## 고찰

간단한 배열 내 값 비교 문제이다. int 배열에 5개의 정수를 입력받고 다시 처음부터 배열을 선형 탐색하면서 MIN과 MAX는 첫 값으로 초기화한 뒤 이후 인덱스와 비교, 평균은 입력 수가 5로 정해졌기 때문에, 탐색한 정수를 각각 탐색 때마다 5로 나눠 avg 변수에 저장하였다.

이후 MIN, MAX, AVG를 한 줄씩 출력하였다.

이 문제에서 소수점 자리 출력을 조절하는 것(fixed, precision)을 배우게 되었다.

2.

## 문제 설명

1줄에 10개 수의 간격을 맞춰 정렬 후 출력하고, 그 줄의 합과 평균 출력을 5번 시행하는 프로그램이다. 이때, 평균은 소수점 첫째 자리에서 반올림한다.

## 알고리즘

---

**Algorithm 1** Fill a 5x10 array with random numbers and print the sum and average of each row

---

```
1: Declare and initialize an integer 2D array arr: size  $5 \times 10$ , all elements 0
2: Declare an integer variable tmp
3: Initialize init with the current time as seed value
4: Set the number of digits after the decimal point to 0
5: Set the fixed-point notation for output
6: Set the seed value as init
7: for  $i \leftarrow 0$  to 4 do                                ▷ Loop for each row
8:   Initialize an integer variable sum to 0 to store the sum of each row
9:   for  $j \leftarrow 0$  to 9 do                                ▷ Loop for each column
10:    Generate a random number between 1 and 99 and store it in tmp
11:    while  $tmp \geq 100$  or  $tmp \leq 0$  do                    ▷ Loop to generate a random
    number between 1 and 99
12:    Generate a random number between 1 and 99 and store it in tmp
13:  end while
14:  Store tmp in  $arr[i][j]$                                 ▷ Store the random number in the array
15:  Set the output width to 2
16:  Print the element  $arr[i][j]$  of the array
17:  if  $j < 9$  then
18:    Print a space between columns
19:  end if
20:  Update sum by adding  $arr[i][j]$ 
21: end for
22: Print a separator between rows, sum, and average: |
23: Set the output width to 3
24: Print sum of the row
25: Set the output width to 2
26: Set a floating-point variable  $avg = sum$  to calculate the average
27: Print the average  $\frac{avg}{10}$ 
28: end for
```

---

## 결과 화면

23	19	19	70	63	2	99	22	75	86		478		48
91	73	41	23	18	88	22	99	26	32		513		51
38	68	4	67	63	66	90	67	75	23		561		56
57	92	5	52	10	40	88	18	76	92		530		53
97	91	44	80	50	56	88	92	49	22		669		67

각 행에 10개의 숫자가 출력되고 | 로 구분하고 합과 평균을 출력했다.

이때, 각 숫자는 2칸의 크기를 가지도록 width와 띄어쓰기를 이용하여 칸을 정렬하였다. 평균은 precision 이후 fixed를 사용하면 자동으로 반올림이 되기에 이를 활용하여 평균을 출력하였다.

## 고찰

C언어와 다르게 C++는 서식 지정자를 이용한 변수와 소수점 조절 출력이 불가능하기 때문에 cout 클래스 안에 있는 여러 내장 함수를 이용하여 정렬하고 소수점 자리를 조절하였다. 이 부분에서 c++의 출력 포맷, 정렬, 서식 등에 대해 많이 알 수 있었다. 또한, C++에서 난수를 생성할 때 형 변환 방법으로 static\_cast로 설정하는 부분도 공부하면서 알게 되어 이 문제에서 여러 부분에 대해 많은 깨달음이 있었다고 생각한다.

3.

## 문제 설명

3번 문제는 10개의 문자를 입력받고 1부터 10의 각 끝자리부터 가운데로 진행하면서 2개씩 출력하는 프로그램이다.

다시 말해, 배열에서 양 끝 인덱스부터 끝값 2개씩 한 줄에 출력하는 프로그램이다,

## 알고리즘

---

### Algorithm 1 Input ten characters and print symmetric pairs

---

- 1: Declare and initialize a character array *arr* of size 10: all elements initialized to null character
  - 2: Print "Enter ten characters: "      ▷ Request user input for 10 characters
  - 3: **for** *i* ← 0 **to** 9 **do**      ▷ Iterate 10 times to input characters
  - 4:     Read character *arr*[*i*] from input
  - 5: **end for**
  - 6: Print "Input characters are:"    ▷ Print message indicating input characters will be displayed
  - 7: **for** *i* ← 0 **to** 4 **do**      ▷ Iterate to print symmetric pairs
  - 8:     Print *arr*[*i*] and *arr*[9 - *i*]      ▷ Print symmetric pairs
  - 9: **end for**
-

## 결과 화면

```
Enter ten characters: q w e r t y u i o p
Input characters are:
q p
w o
e i
r u
t y
```

10개의 입력을 받고, 양 끝부터 2개씩 띄어쓰기로 구분하여 출력하였다.

## 고찰

C언어 배열에서 배운 수준의 간단한 출력문이었다. 각 행의 인덱스의 합이 9인 것을 생각하며 출력하였다. ( $i + (9-i) = 9$ ) 식을 생각하여 각각의 인덱스를 참조하여 출력하도록 만들었다.

4.

## 문제 설명

4번 문제는 자연수를 입력받아 반대로 출력하는 프로그램이다. 예시로, 1234면 4321, 5670이면 765로 반대로 출력한다. 이때, 0765가 아닌, 765로 출력해야 한다. 처음 수가 0이 오지 않도록 출력하는 것이다.

## 알고리즘

---

**Algorithm 1** Reverse a Number and Print

---

```
1: Initialize variables:  $input, len = 0, tmp, div = 1, flag = 0$ 
2: Declare and initialize an integer array  $reverse$  of size 10: all elements initialized to 0
3: Print "Enter the number: "  $\triangleright$  Request user input for a number
4: Read  $input$  from input  $\triangleright$  Read the input number
5: Set  $tmp \leftarrow input$   $\triangleright$  Store input number in a temporary variable
6: while  $tmp \neq 0$  do  $\triangleright$  Calculate the number of digits in the input number
7:    $tmp \leftarrow tmp \div 10$ 
8:    $len \leftarrow len + 1$ 
9: end while
10: for  $i \leftarrow 0$  to  $len - 1$  do  $\triangleright$  Reverse the input number and store in an array
11:    $reverse[i] \leftarrow (input \div div) \% 10$ 
12:    $div \leftarrow div \times 10$ 
13: end for
14: Print "Reversed number: "  $\triangleright$  Print message indicating reversed number will be displayed
15: for  $i \leftarrow 0$  to  $len - 1$  do  $\triangleright$  Print the reversed number
16:   if  $flag = 0$  and  $reverse[i] = 0$  then  $\triangleright$  Skip leading zeros
17:     Continue to next iteration
18:   else
19:     Print  $reverse[i]$   $\triangleright$  Print the digit
20:      $flag \leftarrow flag + 1$   $\triangleright$  Increment flag when printing the first digit
21:   end if
22: end for
```

---

## 결과 화면

```
Enter the number: 12537000
Reversed number: 73521
```

0은 출력되지 않고 나머지 수가 반대로 출력된다.

## 고찰

입력한 수의 자릿수를 구하기 위해 float 값을 int로 대입할 때 소수점이 사라지는 형 변환을 이용하여 자릿수 값을 구하였다. 이후 자리수에 반대로 저장한 뒤, 처음에 올 0들을 제거하고 숫자를 출력하도록 제어하여 출력하였다.

5.

## 문제 설명

5번은 0부터 32767 사이의 값을 입력받고, 각 자리를 두칸씩 띄어 출력, 줄마다 앞자리 숫자를 지우면서 출력을 반복하는 프로그램이다.

5자리-1자리 순으로 출력하고, 자릿수가 모자란 숫자들은 0을 채운다.

## 알고리즘

---

**Algorithm 1** Reverse a Number and Print

---

```
1: Initialize variables:  $input, len = 0, tmp, div = 1, flag = 0$ 
2: Declare and initialize an integer array reverse of size 10: all elements initialized to 0
3: Print "Enter the number: "  $\triangleright$  Request user input for a number
4: Read input from input  $\triangleright$  Read the input number
5: Set tmp to input  $\triangleright$  Store input number in a temporary variable
6: while  $tmp \neq 0$  do  $\triangleright$  Calculate the number of digits in the input number
7:    $tmp \leftarrow tmp \div 10$ 
8:    $len \leftarrow len + 1$ 
9: end while
10: for  $i \leftarrow 0$  to  $len - 1$  do  $\triangleright$  Reverse the input number and store in an array
11:    $reverse[i] \leftarrow (input \div div) \% 10$ 
12:    $div \leftarrow div \times 10$ 
13: end for
14: Print "Reversed number: "  $\triangleright$  Print message indicating reversed number will be displayed
15: for  $i \leftarrow 0$  to  $len - 1$  do  $\triangleright$  Print the reversed number
16:   if  $flag = 0$  and  $reverse[i] = 0$  then  $\triangleright$  Skip leading zeros
17:     Continue to next iteration
18:   else
19:     Print  $reverse[i]$   $\triangleright$  Print the digit
20:      $flag \leftarrow flag + 1$   $\triangleright$  Increment flag when printing the first digit
21:   end if
22: end for
```

---

## 결과 화면

```
Enter the number: 1245
0  1  2  4  5
1  2  4  5
2  4  5
4  5
5
```

5자리가 아닌 4자리인 수를 입력하여 나머지 자리가 0으로 들어간 화면이다.  
이때, 3자리, 2자리, 1자리인 수를 입력하면 빈 나머지 자리는 0으로 들어가고 채워진 자리의 수가 출력된다.

## 고찰

4번 문제의 각 자리 저장을 활용하여 문제를 쉽게 해결할 수 있었다.

2중 loop에서 내부 loop 변수를 외부 loop 변수로 설정하여 인덱스 참조하는 것이 중요한 문제였다고 생각한다.

또한, 배열을 0으로 초기화하여 4, 3, 2, 1자리가 입력일 때 처리하는 것이 중요하다고 생각했다. (4321, 378, 36, 5 등)

6.

## 문제 설명

6번은 천장함수, 바닥함수, 반올림 함수(사사오입)를 만들어 입력에 대한 세 함수의 함숫값을 출력하는 프로그램이다. 소수점 둘째 자리까지 보이게 해야 한다. 또한, 소수 셋째 자리 이후 부분을 비교하여 천장함수에서 정확히 0이 아니면 소수점 둘째 자리를 올림 해야 하고, 반올림 함수에 입력이 음수일 경우, 절댓값을 기준으로 반올림 해야 한다.

## 알고리즘

주요 과정: 입력 -> 소수점 자리 조정 -> 천장함수 실행 및 출력 -> 바닥함수 실행 및 출력 -> 반올림 함수 실행 및 출력

### 1. 천장함수

입력값을 소수점 둘째 자리까지 정수화한다.

정수부를 다른 변수에 저장한다.

정수화한 변수와 정수부를 빼서 비교한다. (실수부 여부)

0이면 그대로, 아니면 정수화한 변수에 1을 더하고 다시 100으로 나눠 반환한다.

음수는 원점 대칭된 함수인 floor 함수를 이용하여 입력값의 음수(-를 곱함)의 함숫값의 음수(-를 곱함)를 반환하도록 한다.

### 2. 바닥함수

입력값을 소수점 둘째 자리까지 정수화한다.

다시 100으로 나눠 반환한다.

음수는 원점 대칭된 함수인 ceiling 함수를 이용하여 입력값의 음수(-를 곱함)의 함숫값의 음수(-를 곱함)를 반환하도록 한다.

### 3. 반올림 함수

반올림 함수는 바닥함수를 대칭 이동하여 만들 수 있기 때문에 0.005 만큼 대칭 이동한 값을 입력값으로 주어 그 함숫값을 반환한다.

반올림은 절댓값 기준으로 사사오입 반올림을 적용하기 때문에, 양수일 때의 값과 비슷하게 입력값의 음수 (-를 곱함)에 0.005를 더한 함숫값의 음수 (-를 곱함)를 반환한다.

## 결과 화면

```
Enter the floating-point number: -1.066
Ceiling: -1.06
Floor: -1.07
Rounding: -1.07
```

음수의 처리에서, 천장함수는 대수적으로 큰 수로, 바닥함수는 대수적으로 작은 수로, 반올림 함수는 대수적으로 -1.065보다 -1.066이 더 작은 수이기 때문에 내려서 -1.07로 내린 결과이다.

## 고찰

이 문제는 함수의 그래프를 이용하여 풀게 되었다. 각각 함수를 소수점 셋째자리에서 올리는 것이 아닌, 소수 셋째 자리 아래의 값의 존재성을 확인하여 소수 둘째 자리에서 올리는 것을 알아야 하는 문제이다. 각 바닥, 천장함수는 음수, 양수가 원점 대칭인 함수이기 때문에 각 함수의 음수 부분은 입력값과 함숫값에 -를 곱해 반환하여 해결하였다. 반올림 함수는 바닥함수의 대칭이동 함수임을 이용하여 해결하게 되었다.



7.

## 문제 설명

7번 문제는 밑과 지수를 입력받으면 밑의 거듭제곱 값을 출력하는 프로그램이다. 첫 번째 입력은 밑, 두 번째 입력은 지수이다.

## 알고리즘

### Function Declaration

```
int power(int base, int exponent); // Declaration of the power function
```

### Main Function

```
int main(void)
{
    int base, exponent; // Declare variables for base and exponent
    cout << "Enter the base: "; // Prompt for base input
    cin >> base;
    cout << "Enter the exponent: "; // Prompt for exponent input
    cin >> exponent;
    cout << "power(" << base << ", " << exponent << "): " << power(base, exponent); ,
    return 0;
}
```

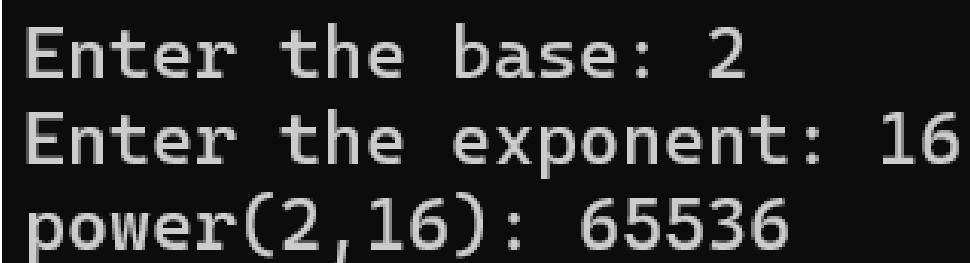
### Function Definition

```
int power(int base, int exponent)
{
    if (!exponent) // When the exponent is 0
        return 1; // Return 1
    else
        return base * power(base, exponent - 1); // Return the exponentiation value
}
```

밑을 입력, 지수를 입력받는다.

이후 재귀 함수의 인자로 두 입력값을 전달하여 0이 될 경우에 1 반환, 아닐 경우에 밑으로 입력받은 수와 재귀 함수 (인자로는 밑과 지수-1인 수를 전달한다)를 반환하여 재귀가 끝난 후 밑의 제곱수로 이루어진 수가 출력되도록 재귀 함수를 작성하였다.

## 결과 화면



2의 16제곱인 65536이 출력된 화면이다.

## 고찰

곱의 재귀 함수 예제로 많이 출제 되어있는 문제이다. 재귀 함수의 반환 값으로 밑과 자신을 곱한 값을 반환하여 재귀가 거듭될수록 곱해지도록 만들었다.

특히 어려운 부분이나 어려운 부분은 없었지만, 밑이나 승수가 음수로 입력될 때의 확장 버전을 만드는 것도, 시도 해 볼 만한 문제라고 생각한다.

8.

## 문제 설명

8번 문제는 두 수를 입력받고, 두 수의 최대공약수를 gcd 함수를 만들어 구하고, 최대공약수를 출력하는 프로그램이다. 이때, 두 수의 크기에 상관없이 출력되게 된다. 둘(입력) 중 하나가 0이면 나머지 하나의 값이 출력된다.

gcd는 Greatest Common Divisor의 약어이다. 최대공약수라는 뜻이다.

## 알고리즘

두 수 입력

gcd함수의 인자로 입력값 전달, 호출

gcd 함수 재귀 호출

gcd 함수 반환값 출력

gcd 함수 설계

두 번째 인자가 0이면 첫 번째 인자 반환

아니면 gcd함수(자기 자신) 호출

이때, 인자는 두 번째 인자와 첫 번째 인자를 두 번째 인자로 나눈 나머지로 전달

## 결과 화면

```
Enter the 1st number: 63
Enter the 2nd number: 1260
gcd(63,1260): 63
```

63과 1260의 최대공약수인 63이 출력된다.

$63=7*3*3$ ,  $1260=2*2*5*7*3*3 \Rightarrow$  최대공약수:  $7*3*3$

## 고찰

이 문제에서 신기한 부분은 입력값의 크기에 상관없이 함수를 인자를 바꿔 호출하고, 나머지 연산을 이용해 호출하는 것만으로도 최대공약수를 구하는 알고리즘을 만들 수 있다는 부분이었다.

$\text{gcd}(b, a\%b)$ 가 가진 의미가 큰 것을 깨닫고 이에 대해 많은 공부가 되었다.

9.

## 문제 설명

9번 문제는 8번에서 구한 최대공약수를 이용하여 최소공배수를 출력하는 프로그램이다

## 알고리즘

두 수 입력

lcm함수의 인자로 입력값 전달, 호출

gcd 함수 재귀 호출

gcd 함수 반환값 lcm함수에 전달, 호출

lcm함수 반환값 출력

gcd 함수 설계

두 번째 인자가 0이면 첫 번째 인자 반환

아니면 gcd함수(자기 자신) 호출

이때, 인자는 두 번째 인자와 첫 번째 인자를 두 번째 인자로 나눈 나머지로 전달

lcm 함수 설계

두 인자를 gcd 함수에 전달, 호출

두 인자의 곱을 gcd 함수의 반환값으로 나눈 값을 반환

$((\text{input1} * \text{input2}) / \text{gcd})$

## 결과 화면

```
Enter the 1st number: 21
Enter the 2nd number: 35
lcm(21,35): 105
```

21과 35의 최대공약수인 7,  $21 \cdot 35 / 7 = 105$ 가 출력됐다.  
이때, 21에 5, 35에 3을 곱한 값이 105이다.

## 고찰

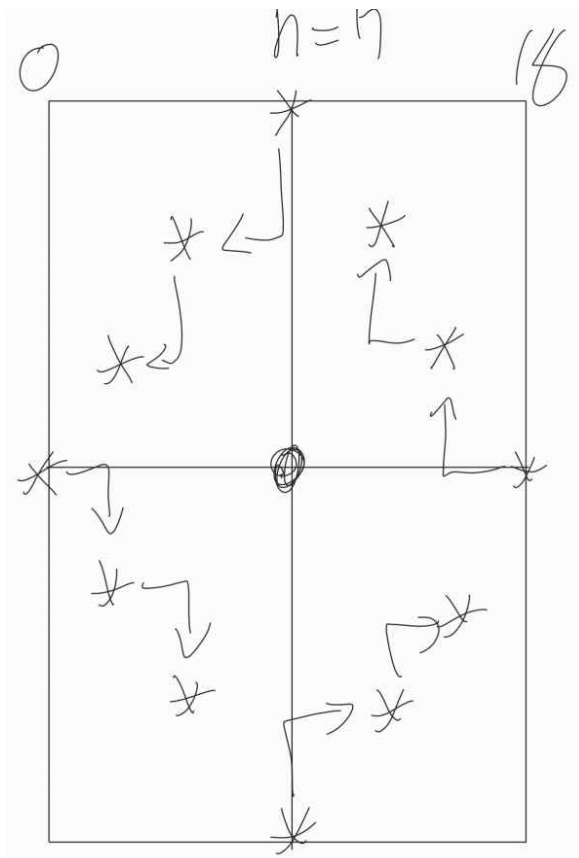
8번 문제인 gcd를 풀었기 때문에 알고리즘에 서술한 수식 하나로 최소공배수를 구할 수 있었다.

10.

## 문제 설명

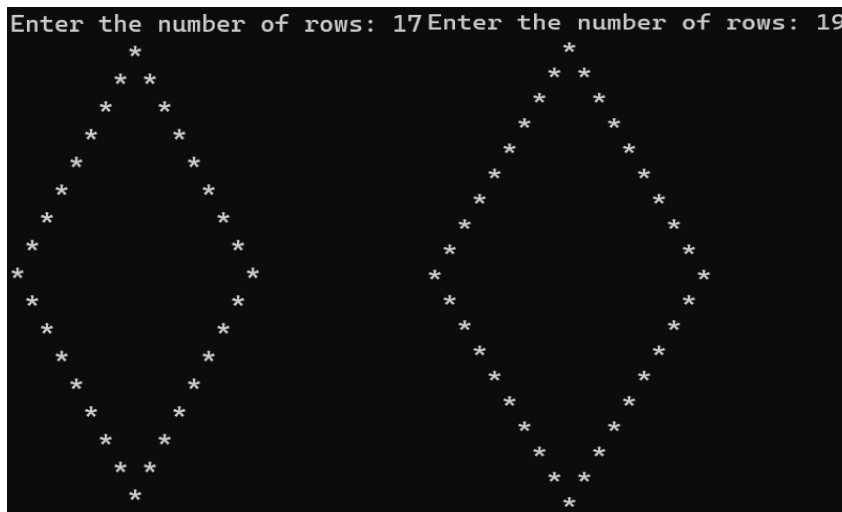
1부터 19까지 홀수를 입력받고 그 길이만큼의 행에 빈 다이아몬드 꼴을 출력하는 프로그램이다.

## 알고리즘



위와 같이 1을 제외한(예외 처리) 입력에 대해, 2차원 배열을 matrix로 생각하여, 가운데 인덱스인 (9,9)를 중심으로 두고 가운데를 포함하여 윗 부분의 별의 개수는 입력값과 같다. 그리고 나머지 별의 개수는 입력값에서 2개만큼 모자란 수와 같다. 이에 따라, 별의 총 개수는  $2n-2$ 개이다. ( $n$ 은 입력) matrix를 4개로 나눠서 계산하면 한 사분면에 들어갈 별의 개수는  $(2n-2)/4=(n-1)/2$ 개이다. 이를 loop에 대입하여 별을 찍으면 +방향 X축과 1사분면 별찍기, +방향 Y축과 2사분면 별찍기, -방향 X축과 3사분면 별찍기, -방향 Y축과 4사분면 별찍기를 통해 별을 입력하고, matrix에서 입력에 대한 행만 출력하도록 했다.

## 결과 화면



17일 경우 17행까지 출력, 19일 경우 19행까지 출력

## 고찰

별찍기 문제는 일반화된 식을 작성하기 어렵다고 알고 있었다. 2차원 배열을 이용하여 별을 찍으니까 많은 식을 세울 필요 없이 하나의 식을 세우고 인덱스 조정을 통해 1중 loop로 문제를 해결할 수 있었다.

만약 행렬이 아닌, loop로만 구현했다면 어땠을지 궁금하기도 하고 이 도형 이외의 문제에 대해 어떤 모형을 만들 수 있을지 연습해보려고 한다.

11.

## 문제 설명

두 번째 입력값이 첫 번째 입력값의 배수인지 판별하는 프로그램이다

## 알고리즘

두 수 입력

multiple 함수에 두 수 차례대로 전달, 호출

multiple 함수

두 번째 인자를 첫 번째 인자로 나눈 나머지가 0인지 확인

맞을 시 true 반환

아닐 시 false 반환

함수의 반환값에 따라 bool 변수가 true면 “true” 출력

false면 “false” 출력

## 결과 화면

```
Enter the 1st number: 6
Enter the 2nd number: 20
multiple(6,20): false
```

```
Enter the 1st number: 4
Enter the 2nd number: 20
multiple(4,20): true
```

20은 4의 배수( $4 \times 5$ )이므로 true를 출력

20은 6의 배수가 아니므로( $6 \times 3 + 2$ ) false를 출력

## 고찰

나머지 연산을 활용하면 문제가 될 것이 없는 간단한 문제였다.

if문 하나로 해결할 수 있는 문제이다.

12.

## 문제 설명

재귀와 반복문을 통해 피보나치 수를 출력하는 프로그램이다. 이때, 편의상 0과 1을 첫 번째, 두 번째 값으로 정의한다.

## 알고리즘

---

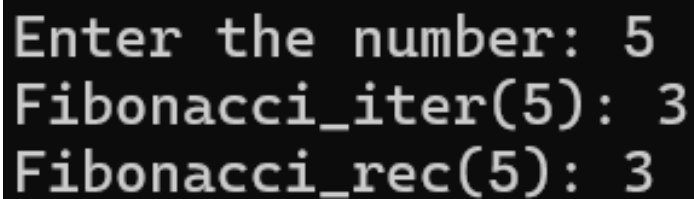
### Algorithm 1 Fibonacci Sequence Calculation

---

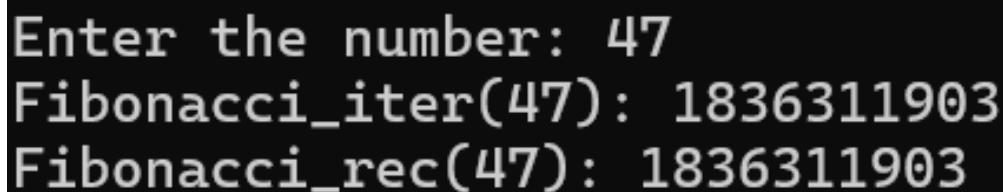
```
1: function FIBONACCI_REC( $n$ )
2:   if  $n = 0$  then
3:     return 0
4:   end if
5:   if  $n = 1$  then
6:     return 1
7:   end if
8:   return FIBONACCI_REC( $n - 1$ ) + FIBONACCI_REC( $n - 2$ )
9: end function
10: function FIBONACCI_ITER( $n$ )
11:    $a[1] \leftarrow 1$ 
12:   for  $i \leftarrow 2$  to  $n - 1$  do
13:      $a[i] \leftarrow a[i - 1] + a[i - 2]$ 
14:   end for
15:   return  $a[n - 1]$ 
16: end function
17: function MAIN
18:    $a[1] \leftarrow 1$ 
19:    $input \leftarrow User\ Input$ 
20:   PRINT("Fibonacci_iter("  $n$  "): ")           ▷ Iterative Fibonacci value
21:   print Fibonacci_iter( $n$ )
22:   PRINT("Fibonacci_rec("  $n$  "): ")           ▷ Recursive Fibonacci value
23:   print Fibonacci_iter( $n$ )
24: end function
```

---

## 결과 화면



```
Enter the number: 5
Fibonacci_iter(5): 3
Fibonacci_rec(5): 3
```



```
Enter the number: 47
Fibonacci_iter(47): 1836311903
Fibonacci_rec(47): 1836311903
```

예시에서 입력값 9가 21이었기 때문에 입력값 5는 3, 입력값 47은 1,836,311,903이 출력된다.

## 고찰

문제에서 첫 번째 값을 0으로, 두 번째 값을 1로 세팅하여, 인덱스가 하나씩 밀리도록 되어있다. 이 부분을 조정하는 문제와 배열의 인덱스 참조에서 액세스 위반(잘못된 주소 참조) 문제를 잘 해결하면 피보나치의 점화식,  $f(n)=f(n-1)+f(n-2)$ 으로 해결할 수 있었다.

13.

## 문제 설명

한 줄의 문자열을 입력받고, 그중에서 소문자는 대문자로, 대문자는 소문자로 변환하는 프로그램을 작성하는 문제이다.

## 알고리즘

### String Case Conversion Algorithm

1. Declare an array *input* to store the string, with a capacity of 101 characters.
2. Prompt the user to enter a string with the message: "Enter the string: ".
3. Read the string entered by the user, up to 100 characters, and store it in the *input* array.
4. For each character in the *input* array, until the null character ('\0') is encountered:
  - (a) If the current character is uppercase (between 'A' and 'Z'), convert it to lowercase by adding 32 to its ASCII value.
  - (b) If the current character is lowercase (between 'a' and 'z'), convert it to uppercase by subtracting 32 from its ASCII value.
5. Output the result by displaying the converted string.



## 결과 화면

```
Enter the string: 2023202032 StuDEnT KWanGWoOn UnIveRsY
Result: 2023202032 sTUdeNt kwANGwOoN uNiVErSy
```

입력된 char형 중, 글자(영문)에 해당하는 student kwangwoon university가 바뀌었다.

중간에 빈칸이 들어가 있어도 getline으로 인해 결과가 제대로 나오게 된다.

## 고찰

이 문제는 C언어에서 아스키코드를 활용하여 많이 해결하던 문제이다.

C++로 바뀌면서 중요한 점은, cin.getline 을 사용하여 띄어쓰기 처리를 하는 부분이 가장 중요한 부분이라고 생각한다.

14.

## 문제 설명

1부터 1000중에서 완전수를 그 합이 덧셈 표현과 함께 각 줄마다 출력하는 프로그램을 작성하는 문제이다.

## 알고리즘

---

**Algorithm 1** Check for Perfect Numbers

---

```
1: function ISPERFECT( $n$ )
2:    $sum \leftarrow 0$ 
3:   for  $i \leftarrow 1$  to  $n - 1$  do
4:     if  $n \bmod i = 0$  then
5:        $sum \leftarrow sum + i$ 
6:     end if
7:   end for
8:   if  $sum = n$  then
9:     return true
10:  else
11:    return false
12:  end if
13: end function
14: main:
15:  $N \leftarrow 1000$ 
16: Print "Perfect numbers between 1 and 1000:"
17: for  $i \leftarrow 1$  to  $N$  do
18:   if ISPERFECT( $i$ ) then
19:     Print  $i$ 
20:   end if
21: end for
```

---

## 결과 화면

```
Perfect numbers between 1 and 1000:  
6 = 1 + 2 + 3  
28 = 1 + 2 + 4 + 7 + 14  
496 = 1 + 2 + 4 + 8 + 16 + 31 + 62 + 124 + 248
```

1부터 1000 사이의 완전수인 6, 28, 496이 그 수의 덧셈의 합과 함께 출력되었다.

```
1 + 2 + 3 =  
6  
1 + 2 + 4 + 7 + 14 =  
28  
1 + 2 + 4 + 8 + 16 + 31 + 62 + 124 + 248 =  
496
```

## 고찰

완전수의 개념을 이해하지 못해, 어떻게 알고리즘을 구성해야 할지 고민을 많이 한 문제이다. 약수들의 합이 자신이 되어야 한다는 점에서 고민했다. 결국 loop 를 이용한 다중 loop로 연산하여 찾아보았다. 이보다 더 좋은 알고리즘은 없는지 궁금해지는 문제였다.

15.

## 문제 설명

숫자 야구 게임이다. 5번의 기회를 주소 같은 자리에 숫자가 맞으면 hit, 숫자가 다른 자리에 있으면 blow를 출력한다.

5번의 기회 안에 4자리의 숫자를 정확히 맞추면 WIN, 5번까지 맞추지 못하면 LOSE를 출력한다.

LOSE 시에는 맞춰야 할 4자리의 숫자를 알려준다.

## 알고리즘

---

### Algorithm 1 Guessing Game

---

```
1: Initialize init with current time using time()
2: Set seed using init
3: Initialize random[4] with out-of-range values
4: Initialize save_index[4] with zeros
5: Initialize input[5]
6: Initialize tmp, hit, blow to zero
7: for i  $\leftarrow$  0 to 3 do
8:   Generate random number tmp between 0 and 9
9:   while duplicate tmp is found do
10:    Generate another random number tmp
11:  end while
12:  Assign tmp to random[i]
13: end for
14: for i  $\leftarrow$  0 to 4 do
15:   Print "Guess:"
16:   Input input
17:   Initialize hit, blow to zero
18:   for j  $\leftarrow$  0 to 3 do
19:    if input[j]  $\neq$  0' = random[j] then
20:      Increment hit
21:      Set save_index[j] to 1
22:    end if
23:  end for
24:  for j  $\leftarrow$  0 to 3 do
25:    for k  $\leftarrow$  0 to 3 do
26:      if (input[j]  $\neq$  0') = random[k] and save_index[j]  $\neq$  1 then
27:        Increment blow
28:      end if
29:    end for
30:  end for
31:  Print "Hit: ", hit, ", Blow: ", blow
32:  if hit = 4 then
33:    Print "Win"
34:    break
35:  end if
36: end for
37: if hit  $\neq$  4 then
38:   Print "Lose"
39:   Print "the correct answer: ", random[0], random[1], random[2],
   random[3]
40: end if
```

---

## 결과화면

```
Guess: 1359
Hit: 0, Blow: 2
-----
Guess: 1825
Hit: 0, Blow: 2
-----
Guess: 1360
Hit: 1, Blow: 1
-----
Guess: 5840
Hit: 1, Blow: 0
-----
Guess: 3827
Hit: 0, Blow: 1
-----
Guess: 3192
Hit: 0, Blow: 1
-----
Lose
the correct answer: 5761

Guess: 9163
Hit: 1, Blow: 1
-----
Guess: 9236
Hit: 1, Blow: 0
-----
Guess: 1636
Hit: 0, Blow: 1
-----
Guess: 9471
Hit: 4, Blow: 0
-----
Win
```

## 고찰

각 자리수의 초기화는 0~9를 벗어난 10으로 초기화 후 모든 자리수와 겹치지 않도록하여 초기화하였다. 게임을 처음 이해했을 때, 5번의 기회 동안 맞춘 횟수를 누적으로 더해서 넘기는 방식인 줄 알고 알고리즘을 이해하지 못했었다. 이후, 4자리 숫자를 맞춘 후 기회를 사용하는 것임을 알고 코드를 작성할 수 있었다. 또한, hit을 blow와 겹치지 않도록 중복 체크하는 배열을 만들어 처리해야 하는 까다로움이 있던 문제였다.