

객체지향프로그래밍 설계 및 실습

ASSIGNMENT1

컴퓨터정보공학부

2017202037 오민혁

설계 : 신 영 주 교수님(월 1,수 2)

실습 : 신 영 주 교수님(목 3,4)

<Assignment1-1>

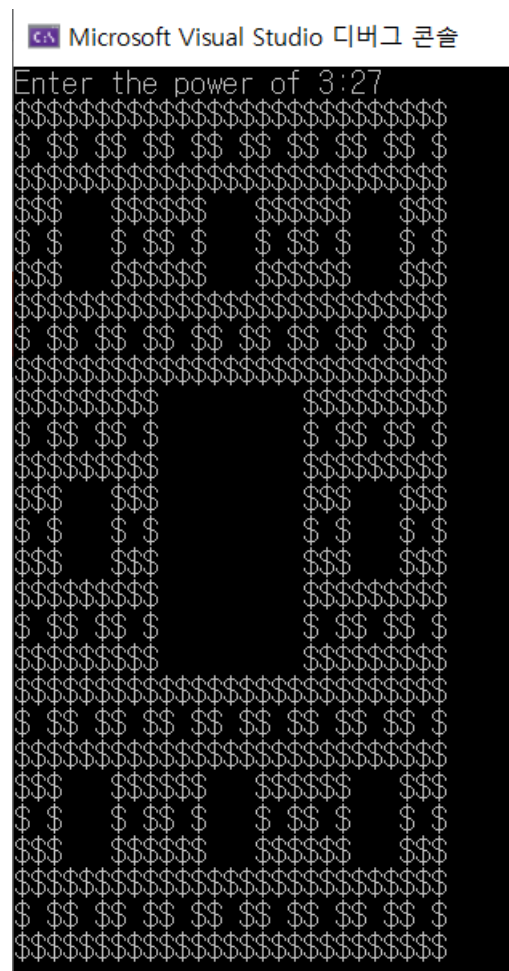
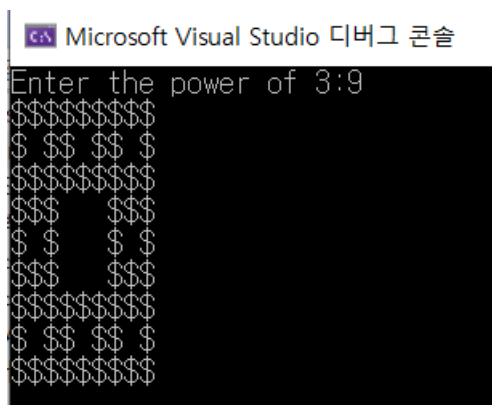
- Introduction

사용자로부터 3 의 제곱에 해당하는 수만을 입력 받아 입력 받은 수를 크기로 지정하여, '\$' 문자로 사각형 프랙탈 모양을 만드는 문제이다.

- Explanation

동적 할당을 이용하여 2 차원 배열을 선언한 후, 프랙탈 구조의 사각형을 그리는 재귀함수를 정의하고, 그 재귀함수를 이용하여 사용자로부터 입력 받은 n 크기만큼 2 차원 배열에 사각형 프랙탈을 그린 후 2 차원 배열 전체를 출력하였다.

- Result Screen



- Consideration

처음에는 2 중 포문을 이용하여 해결하려고 마음을 먹고 공백이 생기는 규칙을 열심히 찾아보았다. 아무리 생각해도 규칙을 찾을 수 없고, 너무 많은 시간을 소비 한 것 같아서 다른 방법을 찾아보는 도중, 이 프랙탈 모양은 같은 모양이 계속 반복된다는 점을 인지하고 재귀함수를 떠올리고 해결할 수 있었다.

<Assignment1-2>


- Introduction

sieve of Eratosthenes 알고리즘을 사용하여 사용자로부터 입력 받은 n 이하의 모든 prime number 를 찾고 출력하는 문제이다.


- Explanation

2 부터 n 까지 반복하는 반복문을 이용하여 bool 형 배열을 생성하여 true 로 초기화 한 다음, n 을 제외한 n 의 배수를 지우는 방식을 계속한다. 반복 문 안에서의 n 의 제공이 사용자로부터 입력 받은 n 보다 커질 때 까지 반복하는 함수를 만들어서 해결하였다.

- Result Screen

 Microsoft Visual Studio 디버그 콘솔

```
Enter the n:20
2 3 5 7 11 13 17 19
The number of Prime numbers:8
```

 Microsoft Visual Studio 디버그 콘솔

```
Enter the n:200
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181
191 193 197 199
The number of Prime numbers:46
```

- Consideration

실습 강의자료에 있는 sieve of Eratosthenes 알고리즘을 참고하여 문제를 해결하였다. bool 형 배열을 이용해야겠다는 생각을 하기까지 시간이 좀 걸렸지만 어렵지 않았다.

<Assignment1-3>

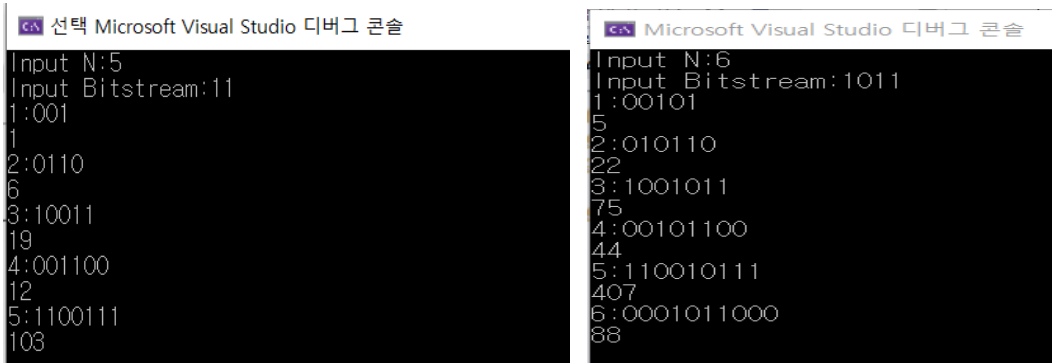
- Introduction

bitwise Not 작동을 하는 $f()$ 함수와 reverse the bit stream 작동을 하는 $g()$ 함수를 구현하여 사용자로부터 입력 받은 n 횟수 만큼 $s_n = \begin{cases} f(g(s_{n-1})) + "1" \\ f(g(s_{n-1})) + "0" \end{cases}$ 에 맞게 2 진수와 10 진수를 출력하는 문제이다.

- Explanation

String 형태로 bitstream 을 입력 받아 이를 인자로 받아서 Not 작용을 하는 f 함수와 revers the bit stream 작동을 하는 g 함수를 구현하였다. 그리고 string 형태의 2 진수를 10 진수로 바꿔서 출력하는 함수도 구현하였다. 사용자로부터 n 과 Bitstream 을 입력 받고 입력 받은 string 형태의 bitstream 을 $f(g(s))$ 형식으로 작업하여 n 이 홀수인지 짝수인지에 따라 맨 끝 비트에 1 혹은 0 을 추가해주는 동작을 n 번 반복하였다.

- Result Screen



```
선택 Microsoft Visual Studio 디버그 콘솔
Input N:5
Input Bitstream:11
1:001
1
2:0110
6
3:10011
19
4:001100
12
5:1100111
103

Microsoft Visual Studio 디버그 콘솔
Input N:6
Input Bitstream:1011
1:00101
5
2:010110
22
3:1001011
75
4:00101100
44
5:110010111
407
6:0001011000
88
```

- Consideration

맨 처음에 $g()$ 함수에 대해 순환 시프트 하는 것으로 잘못 이해하여, 많이 고생하였다. 해결하고자 하는 문제에 대한 정확한 이해가 중요하다는 것을 다시 한 번 깨닫게 되었다. 이 문제를 해결하면서 10 진수를 2 진수로, 2 진수를 10 진수로 바꾸는 방법을 생각해내는게 가장 까다로웠다.

<Assignment1-4>

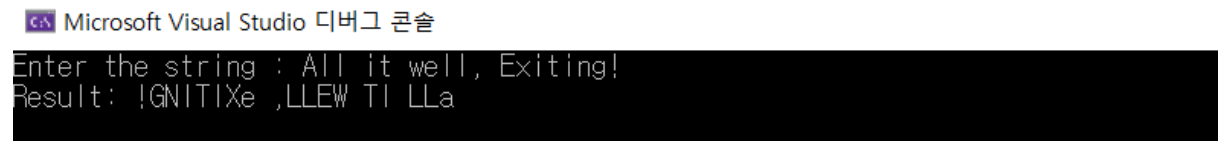
- Introduction

사용자로부터 문자열을 입력 받아 소문자는 대문자로, 대문자는 소문자로 바꾸고 문자열을 전체를 다시 역순으로 출력시켜야 하는 문제이다.

- Explanation

실습 강의자료를 참고하여 CalcLength 함수를 구현하고, +32, -32 를 하면 대,소문자가 바뀌는 점을 이용하여 대소문자를 바꾸었고, 3 개의 변수를 이용하여 배열에 저장된 정보를 바꾸는 알고리즘을 사용하여 문자열의 순서를 뒤바꿔주었다.

- Result Screen



```
Microsoft Visual Studio 디버그 콘솔
Enter the string : All it well, Exiting!
Result: !GNITIXe ,LLEW TI LLa
```

- Consideration

이 문제는 아스키 코드 값을 이용하여 +32 혹은 -32 를 하면 소문자에서 대문자, 대문자에서 소문자로 바뀐다는 특징을 이용하여 쉽게 해결할 수 있었다.

<Assignment1-5>

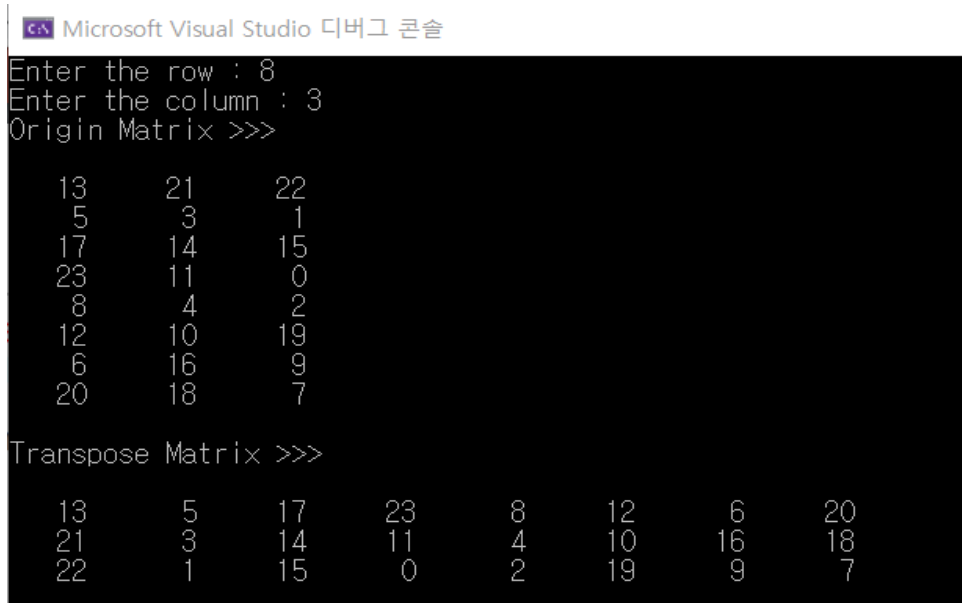
- Introduction

사용자로부터 row, column 을 입력 받고 그에 해당하는 2 차원 배열을 만들어 중복되지 않는 난수를 집어넣고 원래의 행렬과 row, column 이 전환된 행렬을 출력하는 문제이다.

- Explanation

동적 할당을 이용하여 2 차원 배열을 만들고, memset 을 이용하여 초기화 시켜준다음, rand 함수를 이용하여 랜덤 된 값들을 배열에 입력하였다. 입력하는 과정에서 Column*Row 사이즈의 1 차원 배열을 만들고 그것을 checking table 로 이용하여 중복된 수를 걸러내는 작업을 통하여 중복되지 않게 배열에 수들을 입력할 수 있었다.

- Result Screen



```
Microsoft Visual Studio 디버그 콘솔
Enter the row : 8
Enter the column : 3
Origin Matrix >>>
13      21      22
5        3        1
17      14      15
23      11      0
8        4        2
12      10      19
6        16      9
20      18      7

Transpose Matrix >>>
13      5      17      23      8      12      6      20
21      3      14      11      4      10      16      18
22      1      15      0       2      19      9       7
```

- Consideration

이 문제를 해결하면서 행렬을 전환시키는 것은 어렵지 않았는데 중복되지 않게 배열에 수를 넣어야 한다는 점을 해결하는게 까다로웠다. Column*Row 사이즈의 1 차원 배열을 만들고 그것을 checking table 로 이용하여 해결할 수 있었다.

<Assignment1-6>

- Introduction

2 명의 사용자로부터 번갈아 가면서 x,y 좌표를 입력 받고 알맞은 board 좌표에 체크한다.
가로, 세로, 대각선으로 1 bingo 가 완성되면 해당 사용자가 승리하는 게임을 만드는 문제이다.

- Explanation

우선 빙고 판을 만드는 2 차원 배열을 만들고, 사용자로부터 입력 받은 좌표를 이용하여 O, X 문자를 알맞은 좌표에 입력되게 하였다. 그리고 빙고 체크를 하기 위한 2 차원 3x3 int 형 배열을 따로 만들어서 빙고를 체크하였다.

- Result Screen

```
Microsoft Visual Studio 디버그 콘솔

[Play 1] Enter your location [x y] : 1 0
X 0 1 2
Y -----
0|0 |0 |X |
  -----
1|  |X |0 |
  -----
2|  |0 |X |
  -----

[Play 2] Enter your location [x y] : 0 2
X 0 1 2
Y -----
0|0 |0 |X |
  -----
1|  |X |0 |
  -----
2|X |0 |X |
  -----

winner is [Player 2]
```

- Consideration

이 문제를 해결하면서 board 를 그리는 것과 그에 알맞은 x, y 좌표를 찾아야 하는 점이 까다로웠다. 어떻게 해야 할지 고민하다가 빙고 검사용 2 차원 배열을 따로 하나 만들어서 해결할 수 있었고, Bingo 검사하는 과정을 구현하는 것은 어렵지 않았다.

<Assignment1-7>

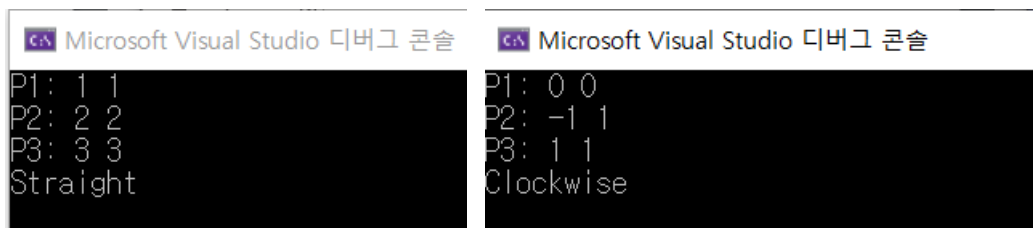
- Introduction

이 문제는 2 차원 평면 좌표 상에서 사용자로부터 3 개의 점 좌표(x, y)를 입력 받아 세 점을 이었을 때의 방향성을 찾아서 출력하는 문제이다.

- Explanation

2 차원 평면좌표에서 3 개의 좌표를 이용하여 p1,p2,p3 좌표를 이어서 외적을 이용한 삼각형의 넓이 구하는 공식을 이용해서, 이 넓이가 0 보다 크면 Counter-clockwise, 0 이랑 같으면 Straight, 0 보다 작으면 Clockwise 가 출력되게 하였다.

- Result Screen



- Consideration

3 개의 좌표를 가지고 벡터의 외적을 이용하여 삼각형의 넓이를 구하는 공식을 이용하여 쉽게 해결 할 수 있었다.

<Assignment1-8>

- Introduction

파일 입출력을 이용하여 text 파일에 있는 16 진수 Caesar cipher 암호를 읽어오고, 이를 다시 10 진수로 바꾼 후, 128 가지의 경우에 수에 따라 각각 하나씩 출력해보고 암호 해독하는 문제이다.

- Explanation

우선 파일 입출력 시스템을 이용하여 ciphertext.txt 에 있는 내용들을 읽어와서, 이를 stringstream 을 이용하여 string token 을 사용해서, 16 진수들을 10 진수로 변환시키면서 배열에 넣어준다. 그리고 128 번 반복하는 for 문을 이용하여 128 가지 경우의 수에 따른 암호들을 출력한다. 이 때 ASCII 코드는 127 까지 밖에 없다는 점을 인지하여 예외처리를 해주어야 하는데 127 을 넘는 숫자는 128 을 128 보다 작아질 때 까지 빼줘야한다.

- Result Screen

```
key(78): =lsjnia[bs&-nb_s-m[s&-cm-nb_-mnlia_mn-fche-ch-nb_-]b[ch(-Mnlia-jl[cm_-ch^__&-#on-cn(m+[fmi-mig_qb(n-^cmgn
mcp_(-C-)]lsjnia[bs&-cm-ch^_]n-nb_-mnlia_mn-j[ln-i-]siol-msmn_g&-qbs-chp_mn-ncg_-cgjlipcha-cn^qb_h-nb_-l-]l_-mi-g[hs-
inb_-]l_-[m-i]-nb_-msmn_g-nb[n-qcff-#_h_-cn-gil-]lig-siol-[nn_hncih9]
key(79): >mtkojibm#kct'e't#t'nc'omjib'nodific'c#di)lNomjibm#da'i_-_'oIngnjih`rc#odnhndndq`)lDamtkojibm#kctni#^oc'omjib'no
#moajpntno'h'etia'nodh'hkmjadibo[rc'ic'm'm'j#itoc'mm'#nac'tno'hc#odggjm'mhjpmoo'iodji:]
key(80): ?nulpkcn]ldu(LpdauLo]u(LeoLpdaLopnkjcaopLhejgLejLpdaL_d]ej*-Opnkjcln]eoalej'aa'(L^qpLep]oL]hokLokiasd]pL'eoieo
oera*-Ebl_nulpkcn]lduLeoLejLb]_pLpdaLopnkjcaopL]nplkbLukqnLouopai(LsduLejraopLpeiaLeiInkrejclEp-sdajLpdanaL]naLokLi]juL
kpdanL]naLoLkbLpdaLouopaiLpd]pLsehhL^ajabepLiknaLbnkiLukqnL]ppajpekj;-
key(81): @ovmqldo'mev)+qebv+p^v)+fp+qeb+pqoIkdbpq+ifkh+fk+qeb+^e'fk+PqoIkdm+mo^fbb+fkabba)+_rq+fq+p+^ipL+pLjbt^a+afpJfpp
fsb+Fc+^ovmqldo'mev+fp+fk+c^q+qeb+pqoIkdbpq+m'oq+lc+vIro+pvpqbj)+tev+fksbpb+qfjb+fjmolfskd+fqtebk+qebob+^ob+pl+j`kv+Iqe
bo+^ob`p+lc+qeb+pvpqbj+qe^a+tfii+^bkbefq+Jlob+colj+vIro+^qabkafIk<
key(82): Apwnrmep_nfw*Arfca_w*AgArfcaArpmleqarAjglIglArfcaaf_gIOrpmleAnp_gacAgIbcbcb*ArAgrDqA_jamAmkuf_rAbgakgagg
tcGaApwnrmep_nfwAgAgIld_arArfcaArpmleqarAn_prAndAmmpaQawqack*ufwAgIteqArgrgkAgknpmntgleAgufcIArfcpca_PcAmak_IwAmrfcpA
_pc_aAndArfcaQawqackArf_rAugjja_cldgrAkmpcAdmkAwmspa_rrelrgml=
key(83): Bqxsfnf qgxVsgdxVr^x+VhrVsgdVrsqnmfdrsvKhmjVhmVsgdVbg^hm- RsqnmfVog^hrdVhmdocd+VatsVhs_rVkrnVrnlDvg^sVchr
lHrrud- HeVbqxsfnf qgxVhrVhmVebsVsgdVrsqnmfdrsvOqVneVxntaVrxrsdl+VvgxVhmudrsVshldVhloqnuhmfVhs_vgdmVsgd
qdVqdrnVl^mxVnsdgaVqd^rVneVsgdVrxrsdlVsg^sVhkkVadmdehsVInqVeqnVxntaVssdmshnm>
key(84): Cryptography, they say, is the strongest link in the chain.
Strong praise indeed, but it's also somewhat dismissive.
If cryptography is in fact the strongest part of your system, why invest time improving it
when there are so many other areas of the system that will benefit more from your attention?
key(85): Dszauphsbaiz-luifz!tbz-!t!uif!tuspohtu!mjol!jo!uif!dibjo/8Tuspoh!qsbjtf!joeffe-!cvu!juu!t!bmtptpnfxibu!ejtnjt
t!jwf/8Jg!dszauphsbaiz!jt!jo!gbdu!uif!tuspohtu!qbsu!pg!zpvstztufn-!xiz!jowftu!ujnf!Jnqspw!oh!ju8xi!fo!uifsf!bsf!tp!nboz!
puifs!bsfbl!pg!uif!tztufn!uibu!xjnm!cfofgju!npsf!gspl!zpvstbuufoujpo88
key(86): Et{rvaitorj{"vJg{"uc{."ku"vJg"vutapiuv"nkpm"kp"vJg"ejckp08Uvtapi"rtckug"kpfggf."dw"kv"u"cnua"uqogyjcv"fkuoku
ukxg08Kh"et{rvaitorj{"ku"kp"hcev"vJg"vutapiuv"rtcv"dh"lqwt"u{uvgo."vj{"kpxguv"vkog"kortaxkpi"kv4yjb"vjgtg"ctg"uq"ocp{"
qvjet"ctgcu"dh"vJg"u{uvgo"vjcv"yknn"dgpgkhv"oatg"htq"lqwt"cvvgpvkqapA8
zkha#wkhu#duh#vr#pda]#rwkhu#duhdv#ri#wkh#v]vwhp#wkd#wzloo#ehqhi]lw#pruh#iurp#lrxu#dwwhqlrqB
key(88): Gv!txskvet!}0$xl!i}$we!0$mw$xl!i}$wxvsrk!wx$pmro$mr$xl!i}$glemr2#Wxvsrk$tvemw!$mrhi!h0$fyx$mx!w$epws$wsqi!lex$hmwamw
```

- Consideration

이 문제를 해결하면서 많은 공부를 하였다. ifstream, ofstream 을 이용한 파일 입출력 시스템과, stringstream 을 통한 string token, 그리고 16 진수를 10 진수로 바꾸는 작업 등, 배울게 많은 문제였다. 많은 공부를 필요로 하는 문제였다. ASCII 코드에 대한 이해가 부족하여 ASCII 코드가 0~127 까지 밖에 없다는 점을 인지할 때 까지 많은 시간이 걸렸다.

<Assignment1-9>

- Introduction

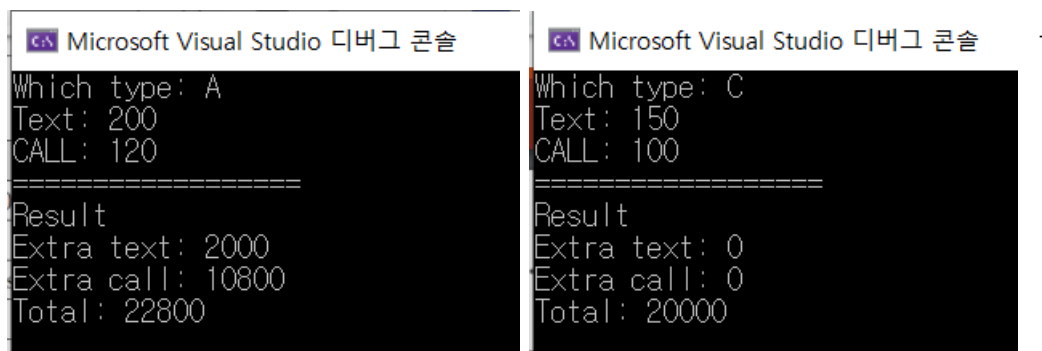
사용자로부터 요금제 명, 요금제의 무료 문자 수, 통화량을 입력 받아 현재 사용하는 요금제의 조건에 따라 초과 문자 사용료, 통화료를 출력하고 총 핸드폰 요금을 계산하여 출력하는 문제이다.

- Explanation

각 요금제에 해당하는 조건들을 요금제명을 조건으로 한 제어문을 4 가지 만들어서

요금과 문자량, 통화량을 곱하는 수식을 통해 초과 요금을 계산하여 요금제와 더하는 방식으로 구현하였다.

- Result Screen



```
Microsoft Visual Studio 디버그 콘솔
Which type: A
Text: 200
CALL: 120
=====
Result
Extra text: 2000
Extra call: 10800
Total: 22800

Microsoft Visual Studio 디버그 콘솔
Which type: C
Text: 150
CALL: 100
=====
Result
Extra text: 0
Extra call: 0
Total: 20000
```

- Consideration

해당 요금제에 따른 조건들을 제어문으로 구현하여 어렵지 않게 해결할 수 있는 문제였다.

<Assignment1-10>

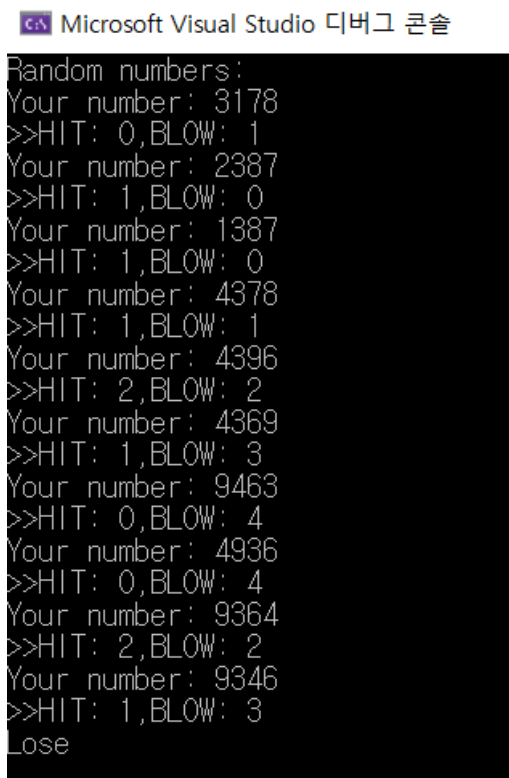
- Introduction

각 자릿수가 중복되지 않는 4 자리 난수를 생성하고, 사용자로부터 똑같이 중복되지 않는 4 자리 수를 입력 받아 난수와 4 자리 수 중에 같은 수가 있으면 BLOW, 근데 위치도 똑같으면 HIT 에 해당하여, 총 4HIT 가 되면 성공하는 Hit & Blow Game 을 만드는 문제이다.

- Explanation

rand함수를 이용하여 4자리 난수를 입력 받은 후 이를 10으로 나누면서 4 크기의 1차원 int 형 배열에 하나 씩 쪼개서 넣어주었다. 그리고 사용자로부터 4 자리 숫자를 입력 받아 이 역시 난수를 쪼개는 방법과 동일하게 1 차원 배열에 넣어주었다. 그 후 반복문과 제어문을 이용하여 숫자의 위치, 같은 숫자가 있는지를 확인하여 Hit 혹은 Blow 변수에 1 씩 더해주며 Hit 가 4 가 될 시에 게임 성공하며, 10 번의 시도에도 Hit 가 4 가 안되면 게임 실패하게 하였다.

- Result Screen



```
Microsoft Visual Studio 디버그 콘솔
Random numbers:
Your number: 3178
>>HIT: 0,BLOW: 1
Your number: 2387
>>HIT: 1,BLOW: 0
Your number: 1387
>>HIT: 1,BLOW: 0
Your number: 4378
>>HIT: 1,BLOW: 1
Your number: 4396
>>HIT: 2,BLOW: 2
Your number: 4369
>>HIT: 1,BLOW: 3
Your number: 9463
>>HIT: 0,BLOW: 4
Your number: 4936
>>HIT: 0,BLOW: 4
Your number: 9364
>>HIT: 2,BLOW: 2
Your number: 9346
>>HIT: 1,BLOW: 3
Lose
```

- Consideration

이 문제를 해결하면서 난수와 사용자로부터 입력 받은 4 자리 수를 비교하기 위해 한 자리 씩 쪼개서 배열에 넣는 방법을 떠올리기 까지 많은 시간이 걸렸다. 그리고 4 자리 각 자릿수의 수들을 따로 보았을 때 중복되지 않게 하는 것을 구현하는 것이 어려웠다.