



Object-Oriented Programming Report

Assignment 1-3

Professor	Donggyu Sim
Department	Computer engineering
Student ID	2022202061
Name	Seoeun Yang
Class (Design / Laboratory)	1 / B (미수강시 0로 표기)
Submission Date	2023. 3. 31

Program 1

□ 문제 설명

3 차원 좌표를 입력 받아 행렬 변환을 수행해주는 프로그램이다. T 연산자는 행렬 T1, T2, T3 를 곱해 만든다. T1 은 z 축에 대한 회전, T2 는 yz 평면에 대한 반사, T3 는 xy 평면에 대한 직교 투영이다. 연산된 T에 입력 받은 좌표를 넣고 3 차원 좌표와 행렬곱 연산을 해주는 프로그램이다.

처음 문제를 풀었을 때 0 이 나오는 결과값이 0 에 근사하게 출력됨을 확인했다. 1-1 과제에 나왔던 것처럼 근사한 두 수의 뺄셈에서 발생하는 부동소수점 오류였다.

이를 방지하기 위해 T에 입력 받은 좌표를 넣어 먼저 연산해준다. 그 후 각 수에 10000 을 곱해주고 round() 연산 후 다시 10000 을 나눠준다. round() 함수는 소수점 첫번째 자리에서 반올림해주는 함수로 반올림한 정수 부분만 반환한다. 이 과정을 거치면 부동소수점 에러가 발생하지 않는다. 이후에 행렬곱 연산을 해주면 알맞은 결과값이 출력된다.

□ 결과 화면

```
Microsoft Visual Studio 디버그 콘솔
9 Degrees : 45
10 Coordinate : 1 1 1
11
12          0      1.4142      0
13 C:\Users\82108\source\repos\객체지향프로그래밍\obj\Debug\
14 연산기.exe (콘솔)
15
```

문제지에 나와있는 예시

```
Microsoft Visual Studio 디버그 콘솔
9 Degrees : 30
10 Coordinate : 1 1 1
11
12      -0.366      1.366      0
13 C:\Users\82108\source\repos\객체지향프로그래밍\obj\Debug\
14 연산기.exe (콘솔)
15
```

고찰

부동소수점 문제를 해결하는 것이 어려웠다. 처음엔 행렬곱까지 계산해주고 반올림을 해줬는데 문제가 해결되지 않았다. 왜냐하면 부동소수점 에러는 행렬곱 계산 이전에 발생하기 때문에 이미 일어난 에러에 행렬곱을 수행해주니 에러가 고쳐지지 않았던 것이다. 또한 10000 을 곱해주고 나누는 과정이 이해가 되지 않았는데 해주지 않을 경우 소수점이 아예 표현되지 않아 정확한 값을 알 수 없다. 이러한 일련의 과정을 통해 문제 설명에 작성한 부동소수점 오류 해결 과정이 완성된 것이다. 연산자 T 를 직접 구하지 않아도 되었던 문제여서 어렵지는 않았던 것 같다.

Program 2

문제 설명

Vs, R1, R2, RL 을 입력 받아 Vout 을 계산하는 프로그램이다. load_power_ratio 는 전원 공급 장치에서 공급되는 전력에 대한 부하 저항에 전달되는 전력의 비율이다. Vout 이 순환소수라면 순환마디를 표현해줘야 한다. 순환마디를 구하는 방법은 다음과 같다.

1. 입력 받은 두 수를 '/' 연산을 해주어 정수 부분을 구한다.
2. 입력 받은 두 수를 '%' 연산을 해주어 나머지 부분을 구한다.
3. 나머지에 10 을 곱해주고 다시 '/' 연산을 해 몫을 저장한다. (이 때의 몫은 순환소수의 수이다.)
4. 순환마디를 찾을 때까지 반복해 주는데 이전에 나왔던 나머지가 나왔다면 그 다음 나머지 연산의 나머지가 이전과 동일하게 나온다.
5. 같은 나머지가 나왔다면 순환소수의 순환마디를 찾음으로 간주하고 출력한다.

$$\begin{array}{r}
 11.4295714 \\
 7 \overline{) 80} \\
 \underline{77} \\
 30 \\
 \underline{28} \\
 20 \\
 \underline{19} \\
 60 \\
 \underline{56} \\
 40 \\
 \underline{35} \\
 50 \\
 \underline{49} \\
 10 \\
 \underline{7} \\
 30
 \end{array}$$

□ 결과 화면

```
Microsoft Visual Studio 디버그 콘솔
Vs : 40
R1 : 4
R2 : 2
R(Load) : 0
Vout : 80/6 = 13.(3)
C:\Users\82108\source\repos\문제지 첫번째 예시
```

Vout = 13.33333333....

```
Microsoft Visual Studio 디버그 콘솔
Vs : 10
R1 : 4
R2 : 2
R(Load) : 2
Vout : 10/5 = 2.(0)
Load power ratio : 10.00%
C:\Users\82108\source\repos\문제지 두번째 예시
```

Vout = 2 (순환소수가 아님)

```
Microsoft Visual Studio 디버그 콘솔
Vs : 40
R1 : 5
R2 : 2
R(Load) : 0
Vout : 80/7 = 11.(428571)
C:\Users\82108\source\repos\문제지 세번째 예시
```

Vout = 11.428571428571428571...

□ 고찰

Vout 을 출력하는 것은 어렵지 않았다. 이 문제의 경우 순환소수에서 순환마디를 구할 수 있느냐가 중점이었던 것 같다. 순환마디를 구하기 위해 특징을 찾던 도중 수를 계속 나눴을 때 나오는 나머지가 같아지면 순환마디가 반복되며 순환소수를 이룬다는 것을 깨닫게 되었다. 따라서 2*100 칸짜리 배열을 저장해 몫과 나머지를 저장해 순환마디를 찾았다. 100 칸짜리로 선언했기 때문에 100 이상의 순환마디를 구할 수 없는 것이 한계다.

Program 3

□ 문제 설명

2,8,10,16 진수를 원하는 진법으로 변환해준다. 어떠한 수를 입력하고 이 수의 진법과 변환하고 싶은 진법을 입력하면 변환된 수를 출력해준다.

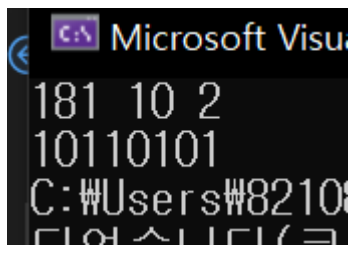
16 진수 때문에 문자열로 입력을 받는다. 문자열의 길이를 파악하고 반복문을 통해 입력 받은 수의 진법을 파악한다. 16 진수가 아니라면 문자열 안의 수를 정수형 배열에 저장해주고 16 진수라면 그대로 문자열을 사용한다. 사용자 정의 함수를 3 개 만들었는데 각각

1. 2,8,10 진수를 int 형 변수에 저장하는 함수
2. 16 진수를 int 형 변수에 저장하는 함수
3. 10 진수를 2,8,16 진수로 변경하는 함수

이다. 함수의 로직은 다음과 같다.

1. 배열 안의 있는 수를 각 진법에 맞게 2,8,10 의 거듭제곱을 곱해 정수형 변수에 더해준다.
2. 알파벳을 그에 상응하는 숫자로 변환하고 정수형 배열에 저장한다. 정수형 배열에 16 의 거듭제곱을 곱해 정수형 변수에 더해준다.
3. 10 진수를 변환하고 싶은 진법에 맞게 나눠주는데 재귀적으로 함수를 호출해 차례대로 나머지를 출력해 표현한다.

□ 결과 화면



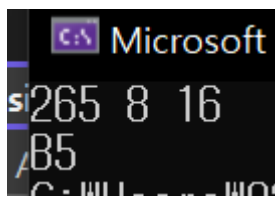
181 10 2
10110101
C:\Users\8210>

$181_{10} \rightarrow 10110101_2$



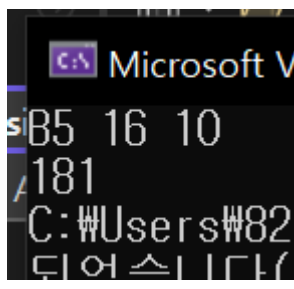
10110101 2 8
265
C:\Users\8210>

$10110101_2 \rightarrow 265_8$



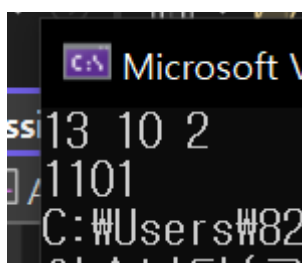
265 8 16
B5
C:\Users\8210>

$265_8 \rightarrow B5_{16}$



B5 16 10
181
C:\Users\8210>

$B5_{16} \rightarrow 181_{10}$



13 10 2
1101
C:\Users\8210>

$13_{10} \rightarrow 1101_2$



C8 16 10
200
C:\Users\8210>

$C8_{16} \rightarrow 200_{10}$

□ 고찰

처음 문제를 풀 땐 각 경우의 수를 다 구현하려고 했다. 예를 들면 2 진수에서 16 진수로, 16 진수에서 8 진수로 변환하는 것처럼 총 12 가지 경우를 모두 구현하려고 했는데 구현하던 도중 결국 어떠한 경우들은 10 진법으로 변환해야 구할 수 있었다. 또한 정수형 변수로 입력 받았었는데 이 경우 16 진수를 입력 받을 수 없다는 문제가 있었다. 조금씩 수정을 해보려고 했는데 도저히 감당할 수 없어서 결국 처음부터 코드를 다시 짰다. 문자열로 입력 받기 때문에 10 진법으로 변환할 때 정수형 배열을 거쳐 변환해야 했고 16 진수의 경우 정수형 배열에 저장할 때 알파벳과 숫자에서 빼줘야하는 아스키코드값이 다를 것을 체크해줘야 한다. 10 진수를 16 진수로 변환할 때는 16 진수 표를 문자열에 저장해 전역 변수로 설정해 각 나머지가 9 이상이라면 그에 맞는 알파벳을 출력하도록 했다.

Program 4

□ 문제 설명

숫자 배열을 입력 받아 오름차순으로 정렬해 출력한다. Bubble sort, Insertion sort, Quick sort, Merge sort 를 통해 정렬하며 각 정렬 방법의 시간 복잡도를 설명한다. 각 정렬 방법의 시간 소요도가 어느 정도인지 측정한다. 숫자 배열은 홀수이며 숫자 배열의 길이를 입력 받아 길이만큼의 배열을 선언해야 한다. 따라서 길이를 입력 받은 후 동적 할당을 통해 정수형 배열을 선언한다. 중간값은 (길이)/2 의 인덱스에 저장된 값을 반환한다. 시간 복잡도의 경우 시간의 흐름으로 파악하려 하였으나 ms 단위로는 파악할 수 없어 반복문의 횟수로 시간 복잡도를 설명하려고 했다. 하지만 Windows.h 헤더파일을 통해 CPU 의 클럭 수의 차이로 시간의 흐름을 파악하면 0.1us 까지 측정할 수 있다. 따라서 반복문의 횟수와 시간의 흐름으로 시간 복잡도를 설명했다.

□ 결과 화면

```
15
26 9 23 19 1 35 4 31 14 13 28 3 18 33 19
Bubble Sort :
Sorted order : 1 3 4 9 13 14 18 19 19 23 26 28 31 33 35
Median number : 19
Number of Repetition : 210
Time spend : 31

Insertion Sort :
Sorted order : 1 3 4 9 13 14 18 19 19 23 26 28 31 33 35
Median number : 19
Number of Repetition : 105
Time spend : 13

Quick Sort :
Sorted order : 1 3 4 9 13 14 18 19 19 23 26 28 31 33 35
Median number : 19
Number of Repetition : 11
Time spend : 13

Merge Sort :
Sorted order : 1 3 4 9 13 14 18 19 19 23 26 28 31 33 35
Median number : 19
Number of Repetition : 14
Time spend : 21
```

문제지 예시

```
Microsoft Visual Studio 디버그 콘솔
23
4 3 2 6 34 32 34 76 45 65 24 29 64 75 56 34 98 908 58 29 27 48 99
Bubble Sort :
Sorted order : 2 3 4 6 24 27 29 29 32 34 34 34 45 48 56 58 64 65 75 76 98 99 908
Median number : 34
Number of Repetition : 506
Time spend : 34

Insertion Sort :
Sorted order : 2 3 4 6 24 27 29 29 32 34 34 34 45 48 56 58 64 65 75 76 98 99 908
Median number : 34
Number of Repetition : 253
Time spend : 21

Quick Sort :
Sorted order : 2 3 4 6 24 27 29 29 32 34 34 34 45 48 56 58 64 65 75 76 98 99 908
Median number : 34
Number of Repetition : 17
Time spend : 21

Merge Sort :
Sorted order : 2 3 4 6 24 27 29 29 32 34 34 34 45 48 56 58 64 65 75 76 98 99 908
Median number : 34
Number of Repetition : 22
Time spend : 24

C:\Users\82108\source\repos\객체지향프로그래밍\1-3\Assignment_4\64\Debug\Assignment_4.exe(프로세스 5872개)이(가) 종료되었습니다.
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록
```

반복 횟수 : bubble > insert > merge > quick

시간 흐름 : bubble > merge > insert = quick


```

Microsoft Visual Studio 디버그 콘솔
13
23 54 86 4 5 4 2 1 76 34 98 45 21
Bubble Sort :
Sorted order : 1 2 4 4 5 21 23 34 45 54 76 86 98
Median number : 23
Number of Repetition : 156
Time spend : 19

Insertion Sort :
Sorted order : 1 2 4 4 5 21 23 34 45 54 76 86 98
Median number : 23
Number of Repetition : 78
Time spend : 11

Quick Sort :
Sorted order : 1 2 4 4 5 21 23 34 45 54 76 86 98
Median number : 23
Number of Repetition : 8
Time spend : 9

Merge Sort :
Sorted order : 1 2 4 4 5 21 23 34 45 54 76 86 98
Median number : 23
Number of Repetition : 12
Time spend : 16

```

반복 횟수 : bubble > insert > merge > quick

시간 흐름 : bubble > merge > insert > quick

□ 고찰

시간의 흐름으로 시간 복잡도를 표현하고 싶어서 `clock()` 함수를 사용해봤는데 정렬 시간이 너무 빠른 나머지 ms 로 시간의 흐름을 확인하려고 해도 0 이 출력되었다. 따라서 CPU 클럭 수를 체크하는 함수를 사용해 0.1us 단위로 시간을 출력했고 각 정렬 방법의 반복문이 몇 번 돌아가는지를 체크해 시간 복잡도를 표현했다.

1. Bubble sort

이 정렬 방법은 배열에서 나란히 붙어있는 수 2 개를 비교해 앞에 위치한 수가 뒤에 위치한 숫자보다 크다면 둘의 순서를 바꿔주는 방법이다. 이중 for 문을 도는데 그 이유는 길이가 N 인 배열이라면 N 개의 수를 정렬해야 하기 때문에 바깥 for 문은 N 만큼 돌고, 그 안의 배열은 숫자 2 개씩 비교해줘야 하기 때문에 N-1 만큼 돈다. 따라서 총 반복문의 횟수는 $N*(N-1)$ 이다. 대수적인 시간 복잡도는 $N(N-1)$ 이지만 두 수를 교환하는 과정에서 temp 변수를 선언해 수를 복사하기 때문에 변수에 저장되어 있는 값을 copy 하는 과정에서도 시간이 흐른다는 것을 기억해야 한다. 따라서 실제 시간 복잡도와는 오차가 있을 것이다.

2. Insertion sort

이 정렬 방법은 배열의 길이, N 만큼 반복문을 돌며 제일 작은 수를 찾아 첫번째 자리부터 순서대로 저장하는 방법이다. 반복문을 돌 때 매번 처음부터 비교하는 것이 아니라 정렬된 만큼을 제외하고 반복하면 되기 때문에 bubble sort 보다 적게 반복한다. 반복 횟수는 $N(N-1)/2$ 으로 이 정렬 방법 또한 제일 작은 수와 위치해야 하는 자리의 수를 바꿔줘야 하기 때문에 temp 변수를 사용한다.

3. Quick sort

이 정렬 방법은 pivot 을 설정해 배열을 정렬한다. pivot 으로 설정한 인덱스의 값을 배열을 탐색해 $arr[a] < arr[pivot] < arr[b]$ 이(가) 되도록 정렬한다. 그 다음 pivot 을 중심으로 배열을 반으로 나눠 앞 배열과 뒤 배열을 정렬한다. 각 배열에서 새로운 pivot 을 설정한다. 이 과정에서 재귀호출이 사용된다. Quick sort 는 입력 받은 배열의 순서가 반복 횟수를 결정하는데 pivot 의 값에 따라 덜 반복할 수도 있고 더 많이 반복할 수도 있다.

4. Merge sort

이 정렬 방법은 배열을 계속 쪼개서 정렬한다. 쪼개고 쪼갬 배열의 크기가 1,2 라면 수의 크기를 파악해 정렬하고 다시 합병한다. 합병하는 과정에서는 새로운 배열을 선언해 두 배열을 계속 비교해 작은 수를 차례대로 저장한다. 배열을 다 쪼개고 정렬하면서 합병해 정렬을 완료한다. 배열을 쪼개기 위해 재귀적으로 호출하기 때문에 반복횟수는 $N-1$ 이다. 배열의 크기가 커진다면 merge sort 의 반복횟수가 기타 다른 정렬 방법보다 적어지지만 매번 함수를 호출하고 배열과 변수를 매개변수로 전달하기 때문에 정렬 시간이 제일 빠르다고는 할 수 없다.

배열을 1000 으로 선언해 숫자 정렬을 시켰을 때, bubble sort 의 CPU 클럭 수가 53187 가 나와 제일 오래 걸리고 merge sort 의 CPU 클럭 수가 902 로 가장 빨랐다. 배열의 크기가 크지 않을 때 merge sort 가 다른 정렬 방법보다 느린 이유는 함수 호출 때문이라고 판단된다. insertion sort 는 main 함수 안에서 반복문을 통해 정렬을 하지만 quick sort, merge sort 는 함수를 호출하고 재귀적으로 호출하기 때문에 시간이 더 걸리는 것으로 파악된다. 하지만 배열의 크기가 충분히 커지면 근본적인 반복문의 횟수가 merge sort 가 적기 때문에 함수를 호출하더라도 다른 정렬 방법보다 빠르게 정렬할 수 있는 것이다.

사실 모든 시간이 us 단위로 출력하는 만큼 시간을 비교해 효율을 따진다는 것 자체가 무의미한 것 같다.

입력

```
Microsoft Visual Studio 디버그 콘솔
1000
361 15 718 513 182 764 585 361 463 689 114 471 620 656 812 630 651 774 237 239 867 559 937 780 574 628 140 168 821 856 330 10 29 898 765 1 251 965 358 112 643 348 812 446 3
58 927 741 352 339 472 683 36 118 963 684 861 871 326 889 715 632 336 654 196 185 583 121 677 560 801 821 445 218 987 940 937 407 712 178 143 144 117 925 534 412 468 965 56
5 856 20 31 123 401 221 440 589 87 433 500 348 773 960 686 777 323 560 148 914 946 564 750 346 423 907 241 403 955 745 313 522 558 578 848 928 56 452 681 955 168 533 306 35
5 76 652 513 432 622 32 443 573 781 353 755 562 830 543 390 400 808 522 964 298 745 17 848 269 721 230 9 19 565 878 766 160 714 91 239 568 371 139 694 402 636 155 701 24 10
7 570 86 981 312 240 570 901 141 973 117 737 204 869 204 181 32 980 329 828 336 976 10 811 293 340 914 539 684 454 79 542 887 238 745 688 593 70 605 568 728 924 346 600 388
7 787 980 620 311 759 890 230 317 281 963 413 46 100 970 747 889 164 763 760 162 340 66 114 653 275 213 989 756 475 530 620 969 984 850 930 506 904 578 632 725 389 268 25 48
7 594 561 122 621 316 763 894 90 801 827 152 814 737 604 475 640 279 335 768 584 312 89 128 700 242 580 522 905 241 389 609 535 986 603 363 884 362 189 315 617 915 637 907
5 304 850 704 514 955 967 431 509 170 312 903 784 447 772 613 401 681 170 992 762 55 116 996 596 967 947 620 461 182 686 666 4 249 370 750 200 191 779 764 676 328 214 305 9
31 395 194 71 649 558 985 734 204 638 640 779 323 445 555 906 491 462 869 702 910 342 130 902 323 544 648 262 957 502 214 650 590 807 960 899 466 590 876 18 402 927 370 988
318 517 955 664 925 331 552 566 582 545 959 533 544 683 647 985 418 554 960 483 428 482 570 318 876 42 159 808 321 242 240 962 159 221 788 77 230 883 717 189 510 324 165 3
56 972 5 981 911 345 366 178 47 638 902 880 478 758 194 849 36 502 328 46 378 489 649 450 942 889 237 703 125 257 76 230 144 916 466 383 794 438 629 774 704 517 576 181 799
382 79 980 730 740 5 56 165 552 80 395 547 102 212 249 186 852 243 462 56 12 428 511 24 186 917 897 887 350 537 151 212 433 948 160 373 472 373 200 83 179 938 544 833 470
50 410 257 103 926 38 22 658 113 555 92 467 96 958 284 574 192 472 991 356 182 120 420 353 203 998 962 171 56 531 859 330 764 749 16 665 337 215 716 657 614 92 899 97 978 2
30 418 893 462 582 890 734 665 929 951 587 278 724 629 813 335 906 747 511 61 941 705 15 81 117 752 438 27 528 198 71 796 1000 99 734 879 646 540 468 231 481 899 284 764 29
5 63 975 745 933 524 214 965 84 203 445 234 689 936 645 761 381 432 534 189 427 145 809 188 705 376 100 40 329 948 287 116 598 160 478 649 164 531 84 698 121 434 797 395 66
699 653 239 713 411 903 433 322 42 270 143 249 63 439 545 536 25 159 446 173 696 440 207 234 66 252 650 515 887 68 136 130 227 786 88 494 855 124 655 250 359 771 94 429 21
2 656 506 942 979 802 238 383 361 531 824 614 618 550 324 182 502 183 903 68 420 643 671 489 859 532 529 897 835 249 318 360 475 304 106 169 722 183 74 217 145 610 419 832
987 497 566 852 370 632 133 107 735 588 698 392 987 610 826 36 482 151 939 903 897 295 455 28 36 555 979 438 894 291 741 81 956 650 722 360 93 61 785 540 801 241 552 454 66
9 7415 222 318 8 189 537 523 39 391 448 671 287 224 407 437 84 533 428 686 49 283 154 289 138 3 791 727 385 183 550 225 393 518 429 659 737 269 122 556 277 663 243 364 839 9
00 62 88 74 193 270 456 127 47 138 862 959 821 6 308 976 641 157 524 355 490 855 84 98 527 424 695 895 616 54 881 742 350 116 500 706 54 868 113 889 954 583 627 204 757 38
434 678 679 795 521 533 524 642 446 781 28 726 114 606 727 179 574 633 251 211 73 861 217 49 629 587 611 561 268 311 137 658 761 437 248 242 503 196 605 448 178 712 613 245
542 979 184 563 971 584 119 383 805 452 926 605 744 402 298 454 681 331 52 224 46 344 579 395 343 140 944 244 208 82 865 618 617 568 859 885 584 849 606 991 691 65 403 61
545 240 54 367 988 418 637 536 903 207 806 90 277 520 636 229 145 239 876 66 797 308 511 82 325 387
Bubble Sort
```

bubble sort

```
988 987 987 971 972 9
8 988 991 991 992 996
Median number : 494
Number of Repetition : 999000
Time spend : 53187
```

insertion sort

```
988 988 991 991 992 99
Median number : 494
Number of Repetition : 499500
Time spend : 11050
```

quick sort

```
8 988 991 991 992 9
Median number : 494
Number of Repetition : 699
Time spend : 709
```

merge sort

```
8 988 991 991 992
Median number : 494
Number of Repetition : 999
Time spend : 902
```