**Programming Tips and Tricks**

자주 까먹는 자잘한 꼼수(?) 및 필수 지식들을 모아놓은 페이지입니다.

Tips, Algorithm

#### 공통

#### Polymorphism(다형성)

Polymorphism

#### 2차원배열 → 1차원배열

```
arr[(i * j.length) + j] //= arr[i][j]
```

#### 단위 변환

### 경위도 변환

```
/* Language : C# */

   /* 도분초 → 천분율 */

        /// <summary>
   /// 도분초 → 천분율
        /// ("128:43:47" → "34:58.105")
        /// </summary>
        /// <param name="dms"></param>
        /// <returns></returns>
        public static string DMS2Permil(string dms)
        {
            return Degree2Permil(DMS2Degree(dms));
        }

        /// <summary>
   /// 도분초 → 천분율
        /// (128, 43, 47 → "34:58.105")
        /// </summary>
        /// <param name="deg"></param>
        /// <param name="min"></param>
```

```
        /// <param name="sec"></param>
        /// <returns></returns>
        public static string DMS2Permil(int deg, int min, int sec)
        {
            return Degree2Permil(DMS2Degree(deg, min, sec));
        }

        /// <summary>
    /// 도분초 → 천분율
        /// ("128:43:47" → {34, 58.105})
        /// </summary>
        /// <param name="dms"></param>
        /// <returns></returns>
        public static Tuple<int, double> DMS2Permil_toTuple(string dms)
        {
            return Degree2Permil_toTuple(DMS2Degree(dms));
        }

        /// <summary>
    /// 도분초 → 천분율
        /// (128, 43, 47 → {34, 58.105})
        /// </summary>
        /// <param name="deg"></param>
        /// <param name="min"></param>
        /// <param name="sec"></param>
        /// <returns></returns>
        public static Tuple<int, double> DMS2Permil_toTuple(int deg, int
min, int sec)
        {
            return Degree2Permil_toTuple(DMS2Degree(deg, min, sec));
        }


    /* 천분율 → 도분초 */

        /// <summary>
    /// 천분율 → 도분초 (string)
        /// ("34:58.105" → "34:58:06")
        /// </summary>
        /// <param name="permil"></param>
        /// <returns></returns>
        public static string Permil2DMS(string permil)
        {
            return Degree2DMS(Permil2Degree(permil));
        }

        /// <summary>
    /// 천분율 → 도분초 (string)
        /// (34, 58.105 → "34:58:06")
        /// </summary>
```

```
        /// <param name="deg"></param>
        /// <param name="sec"></param>
        /// <returns></returns>
        public static string Permil2DMS(int deg, double sec)
        {
            return Degree2DMS(Permil2Degree(deg, sec));
        }

        /// <summary>
    /// 천분율 → 도분초 (int[])
        /// ("34:58.105" → {34, 58, 06})
        /// </summary>
        /// <param name="permil"></param>
        /// <returns></returns>
        public static int[] Permil2DMS_toArray(string permil)
        {
            return Degree2DMS_toArray(Permil2Degree(permil));
        }

        /// <summary>
    /// 천분율 → 도분초 (int[])
        /// (34, 58.105 → {34, 58, 06})
        /// </summary>
        /// <param name="permil"></param>
        /// <returns></returns>
        public static int[] Permil2DMS_toArray(int deg, double sec)
        {
            return Degree2DMS_toArray(Permil2Degree(deg, sec));
        }


    /* 경위도 → 천분율 */

        /// <summary>
    /// 경위도 → 천분율 (string)
        /// (128.7297335 → 128:43.784)
        /// </summary>
        /// <param name="latLng"></param>
        /// <returns></returns>
        public static string Degree2Permil(double latLng)
        {
            return ((int)latLng).ToString("00") + ":" + ((latLng -
(int)latLng) * 60).ToString("00.000");
        }

        /// <summary>
    /// 경위도 → 천분율 (int, double)
        /// (128.7297335 → {128, 43.784})
        /// </summary>
        /// <param name="latLng"></param>
        /// <returns></returns>
```

```
        public static Tuple<int, double> Degree2Permil_toTuple(double
latLng)
        {
            return Tuple.Create<int, double>((int)latLng, Math.Round((latLng
- (int)latLng) * 60, 3));
        }


   /* 천분율 → 경위도 */

        /// <summary>
   /// 천분율 → 경위도 (double)
        /// ("128:43.784" → 128.729733333333)
        /// </summary>
        /// <param name="permil"></param>
        /// <returns></returns>
        public static double Permil2Degree(string permil)
        {
            int colonIndex = permil.IndexOf(':');

            int deg = int.Parse(permil.Substring(0, colonIndex));
            double min = double.Parse(permil.Substring(colonIndex + 1,
permil.Length - (colonIndex + 1)));

            return (double)deg + min / 60;
        }

        /// <summary>
   /// 천분율 → 경위도 (double)
        /// (128, 43.784 → 128.729733333333)
        /// </summary>
        /// <param name="deg"></param>
        /// <param name="min"></param>
        /// <returns></returns>
        public static double Permil2Degree(int deg, double min)
        {
            return (double)deg + min / 60;
        }


   /* 경위도 → 도분초 */

        /// <summary>
   /// 경위도 → 도분초 (string)
        /// (128.7297335 → 128:43:470)
        /// </summary>
        /// <param name="latLng"></param>
        /// <returns></returns>
        public static string Degree2DMS(double latLng)
        {
```

```
            int deg = (int)latLng;

            double decimalPart1 = latLng - deg;
            int min = (int)(decimalPart1 * 60);

            double decimalPart2 = (decimalPart1 * 60) - min;
            int sec = (int)(decimalPart2 * 60);

            return string.Format("{0:D02}:{1:D02}:{2:D03}", deg, min, sec);
        }

        /// <summary>
    /// 경위도 → 도분초 (int[])
        /// (128.7297335 → {128, 43, 47})
        /// </summary>
        /// <param name="latLng"></param>
        /// <returns></returns>
        public static int[] Degree2DMS_toArray(double latLng)
        {
            int deg = (int)latLng;

            double decimalPart1 = latLng - deg;
            int min = (int)(decimalPart1 * 60);

            double decimalPart2 = (decimalPart1 * 60) - min;
            int sec = (int)(decimalPart2 * 60);

            return new int[3] { deg, min, sec };
        }


    /* 도분초 → 경위도 */

        /// <summary>
    /// 도분초 → 경위도
        /// ("128:43:47" → 128.72972222222222)
        /// </summary>
        /// <param name="dms"></param>
        /// <returns></returns>
        public static double DMS2Degree(string dms)
        {
            int colonIndex1 = dms.IndexOf(':');
            int colonIndex2 = dms.LastIndexOf(':');

            string strDeg = dms.Substring(0, colonIndex1);
            string strMin = dms.Substring(colonIndex1 + 1, colonIndex2 -
(colonIndex1 + 1));
            string strSec = dms.Substring(colonIndex2 + 1, dms.Length -
(colonIndex2 + 1));

            return double.Parse(strDeg) + ((double)double.Parse(strMin) /
```

```
60) + ((double)double.Parse(strSec) / 3600);
        }

        /// <summary>
   /// 도분초 → 경위도
        /// (128, 43, 47 → 128.72972222222222)
        /// </summary>
        /// <param name="deg"></param>
        /// <param name="min"></param>
        /// <param name="sec"></param>
        /// <returns></returns>
        public static double DMS2Degree(int deg, int min, int sec)
        {
            return (double)deg + ((double)min / 60) + ((double)sec / 3600);
        }
```

### 거리, 속도, 주파수 변환

```
   /* 거리 계산식 */

        /// <summary>
        /// Yard → Meter
        /// </summary>
        public static double YD2M(double yd)
        {
            return yd * 0.91440183;
        }

        /// <summary>
        /// Meter → Yard
        /// </summary>
        /// <param name="meter"></param>
        /// <returns></returns>
        public static double M2YD(double m)
        {
            return m / 0.91440183;
        }


        /// <summary>
        /// Kiloyard → Meter
        /// </summary>
        /// <param name="yard"></param>
        /// <returns></returns>
        public static double KYD2M(double yd)
        {
```

```
            return 1000.0 * YD2M(yd);
        }

        /// <summary>
        /// Meter → Kiloyard
        /// </summary>
        /// <param name="meter"></param>
        /// <returns></returns>
        public static double M2KYD(double m)
        {
            return M2YD(m) / 1000.0;
        }


        /// <summary>
        /// Kiloyard → Hectometer
        /// </summary>
        /// <param name="yard"></param>
        /// <returns></returns>
        public static double KYD2HM(double yd)
        {
            return 10.0 * YD2M(yd);
        }

        /// <summary>
        /// Hectometer → Kiloyard
        /// </summary>
        /// <param name="meter"></param>
        /// <returns></returns>
        public static double HM2KYD(double m)
        {
            return M2YD(m) / 10.0;
        }


        /// <summary>
        /// Kiloyard → Kilometer
        /// </summary>
        /// <param name="yard"></param>
        /// <returns></returns>
        public static double KYD2KM(double yd)
        {
            return YD2M(yd);
        }

        /// <summary>
        /// Kilometer → Kiloyard
        /// </summary>
        /// <param name="meter"></param>
        /// <returns></returns>
        public static double KM2KYD(double m)
```

```
        {
            return M2YD(m);
        }


        /// <summary>
        /// Kiloyard → Nautical Mile
        /// </summary>
        /// <param name="yard"></param>
        /// <returns></returns>
        public static double KYD2NM(double kyd)
        {
            return kyd * 0.493737;
        }

        /// <summary>
        /// Nautical Mile → Kiloyard
        /// </summary>
        /// <param name="meter"></param>
        /// <returns></returns>
        public static double NM2KYD(double nm)
        {
            return nm * 2.02537183;
        }


    /* 심도 계산식 */

        /// <summary>
        /// Meter → Feet
        /// </summary>
        /// <param name="meter"></param>
        /// <returns></returns>
        public static double M2FT(double m)
        {
            return m * 3.280839895013123359580052493438 3;
        }

        /// <summary>
        /// Feet → Meter
        /// </summary>
        /// <param name="feet"></param>
        /// <returns></returns>
        public static double FT2M(double ft)
        {
            return ft * 0.3048;
        }


    /* 속도 계산식 */
```

```
        /// <summary>
        /// Knots → Meter Per Seconds
        /// </summary>
        /// <param name="knots"></param>
        /// <returns></returns>
        public static double KTS2MS(double kts)
        {
            return kts * 0.514444444444444444444444444444444;
        }

        /// <summary>
        /// Meter Per Seconds → Knots
        /// </summary>
        /// <param name="ms"></param>
        /// <returns></returns>
        public static double MS2KTS(double ms)
        {
            return ms * 1.943844492440604751619870410367 2;
        }


    /* 주파수 계산식 */

        /// <summary>
        /// Hertz → Revolutions Per Minute
        /// </summary>
        /// <param name="hertz"></param>
        /// <returns></returns>
        public static double HZ2RPM(double hz)
        {
            return hz * 60.0;
        }

        /// <summary>
        /// Revolutions Per Minute → Hertz
        /// </summary>
        /// <param name="rpm"></param>
        /// <returns></returns>
        public static double RPM2HZ(double rpm)
        {
            return rpm / 60.0;
        }

        public static double M2KM(double m)
        {
            return m / 1000;
        }

        public static double KM2M(double km)
        {
            return km * 1000;
```

```
        }

        public static double M2HM(double m)
        {
            return m / 100;
        }

        public static double HM2M(double hm)
        {
            return hm * 100;
        }

        public static double HM2KM(double hm)
        {
            return hm / 10;
        }

        public static double KM2HM(double km)
        {
            return km * 10;
        }
        /// <summary>
        /// Meter → Nautical Mile
        /// </summary>
        /// <param name="m"></param>
        /// <returns></returns>
        public static double M2NM(double m)
        {
            return KYD2NM(M2KYD(m));
        }
```

#### 재귀(Recursion)

Recursion

#### 정렬 알고리즘(Sort Algorithm)

### 합병 정렬(Merge Sort)

합병 정렬(Merge Sort)

### 중복되지 않는 난수 생성

```
//language : Java
```

```
//1~5 중복되지 않는 난수 발생

int[] arr = new int[5];
Random random = new Random();

for(int i=0 ; i<arr.length ; i++) {
    arr[i] = random.nextInt(arr.length)+1;

    for(int j=0 ; j<i ; j++) {
        if(j!=i && arr[j] == arr[i])
            i--;
    }
}
```

**Java & C#**

**C# - Lambda 이용하여 오름차순 정렬**

```
List<Point> points = new List<Points>();
points.Add(new Point(11.0, 13.0));
//...

points.Sort((Point point1, Point point2) => point1.Y.CompareTo(point2.Y));
```

**포인터와 구조체 기초 개념 예제**

```
#include <stdio.h>
#include <string.h>
#include "stdlib.h"

struct student
{
    char name[20+1];
    int year;
    double score;
};

typedef struct MyStruct
{
    char str[10];
} hyuk_struct;

void main()
{
```

```
 /* 구조체 1*/
    //struct student st_lee = {"Lee",2008,95.4};
    struct student st_lee = {0};
    strcpy(st_lee.name, "HYUK");
    st_lee.year=2008;
    st_lee.score=99.7;

    printf("%s %d %f\n",st_lee.name,st_lee.year,st_lee.score);

 /*구조체 2 - 타입 정의*/
    hyuk_struct asdf = {"Lucas"};

    printf("%s\n",asdf.str);

    printf("\n\n");

 /* 포인터 */
    float val1 = 5.23f;
    float* ptr1 = NULL;
    ptr1 = &val1; // 포인터이기에 주소값만 대입 가능!

    printf("%d\n",ptr1); // val1 주소값 출력
    printf("%f\n",val1);


    printf("\n");

    char val2 = 'A';
    char* ptr2 = &val2;
    *ptr2 = 'X'; // 포인터로 메모리 직접 접근하여 값 바꿈
    val2='Y'; // 변수에 그냥 대입

    printf("%d\n",ptr2);
    printf("%c\n",val2); // 결과적으로 Y 출력

    printf("\n");

 /* 동적 메모리 할당 */
    int *ptr3 = NULL;
    int mem_size = 100;
    ptr3=(int*)malloc(sizeof(int)*mem_size);

    if(ptr3!=NULL)
    {
        memset(ptr3, 0, sizeof(int)*mem_size); // 메모리의 쓰레기값을 모두 0으로 초
기화
        printf("정수 입력 : ");
        scanf("%d", ptr3); // ptr3가 포인터 변수이므로 인자값도 "&"가 아닌 "%"!!
        printf("%d \n", *ptr3); // 주소값이 아닌 실제 참조값을 출력해야하기에
```

```
        free(ptr3); // 해제
    }
    else
        printf("ptr3==NULL\n");

    printf("\n");

  /* 포인터의 포인터 */ //2차원 이상의 다차원 배열에서 쓰인다.
    int iVal = 777;
    int *iPtr = &iVal;
    int **iPPtr = &iPtr;

    printf("%d %d \n",*iPtr, iPtr); // iVal, iVal 주소값
    printf("%d %d %d \n",**iPPtr, *iPPtr, iPPtr); // iVal, iVal 주소값, iVal
주소의 주소값

    printf("\n");

  /* 동적 다차원 배열 */ //2차원배열은 "1차원 배열들의 배열"
    int row=3, col=4;

    int **iPPtrArr=NULL;
    iPPtrArr=(int**)malloc(sizeof(int*)*row); // 가로(행) 수만큼 메모리 할당(확보)

    for(int i=0 ; i<row ; i++) // row수만큼 끊어가며 메모리 할당 → 2차원 배열 만들기
    {
        iPPtrArr[i] = (int*) malloc(sizeof(int)*col); // 인덱스마다 메모리 할당
        memset(iPPtrArr[i], 0, sizeof(int)*col); // 쓰레기값을 0으로 변환
    }

    *(*(iPPtrArr+2)+3) = 111; // iPPtrArr[2][3] = 777
    printf("%d %d %d\n", **iPPtrArr, *(*(iPPtrArr+1)+1), *(*(iPPtrArr+2)+3)
); // 0, 0, 111

    free(iPPtrArr);

    printf("\n");

  /* 구조체 포인터 */
    hyuk_struct qwerty;
    hyuk_struct *qwertyPtr=NULL;
    qwertyPtr=&qwerty;

    strcpy((*qwertyPtr).str, "XXX");
    printf("%s\n",qwertyPtr->str); // 위와 같은 표현
}
```

**c**

**동적 배열 할당**

```
//1차원
int *input;
int staticSize = 할당크기;
input = (int*)malloc(sizeof(int)*staticSize);

//2차원
int **input;
int staticSizeOfROW = 할당크기;
int staticSizeOfColumn = 할당크기;
input = (int(*)[])malloc (sizeof(int*)*staticSizeOfROW);  //행
input[0] = (int*)malloc(sizeof(int*)*staticSizeOfColumn); //열
```

**Android**

**Parcelable**

- Intent를 이용하여 **Reference Type**의 데이터를 넘길 때 사용.

http://arsviator.tistory.com/169

**Uri에서 실제 경로를 받아오는 Method**

- SD Card에 저장된 그림 파일을 ImageView에 붙일 필요가 있을 때 사용.

```
private String getRealImagePath (Uri uri) {
    Cursor cursor = getContentResolver().query(uri, null, null, null, null);
    cursor.moveToFirst();
    int index = cursor.getColumnIndex(MediaStore.Images.ImageColumns.DATA);
    return cursor.getString(index);
}
```

From:
https://ledyx.xyz/ - **Ledyx WIKI**

Permanent link:
**https://ledyx.xyz/doku.php?id=programming_tips_and_tricks&rev=1442198709**

Last update: **2026/10/24 07:03**