# Example 2
# Using 2-d Arrays

*Young W. Lim*

December 13, 2017

# 1 A spreadsheet example using 2-d arrays

[1]

- `int X[4][SIZE];`
  - `X[0][SIZE];` student ID (identification) number
  - `X[1][SIZE];` Korean subject score
  - `X[2][SIZE];` English subject score
  - `X[3][SIZE];` Mathematics subject score
- `double A[SIZE];` average score of three subjects

## 1.1 avg3() function

```
//--------------------------------------------------
// Calculating the average of three numbers
//--------------------------------------------------
double avg3(int x, int y, int z)
{
  return (x+y+z) / 3.;
}
```

- takes three integers
- returns their arithmetic average value (a double type)

## 1.2 init_arrays() function

```
//--------------------------------------------------
// Initialize X[4][SIZE] arrays
// by assigning random number grade
//--------------------------------------------------
void init_arrays (int X[][SIZE], double A[])
{
  int i;

  // srand(7) makes rand() generate
  // the same random sequence
  // --> easy to debug a program
  srand(7);

  for (i=0; i<SIZE; ++i) {
    X[0][i] = i+1 + 201600; // I
    X[1][i] = rand() % 101; // K
    X[2][i] = rand() % 101; // E
    X[3][i] = rand() % 101; // M
    A[i] = avg3(X[1][i], X[2][i], X[3][i]);
  }
}
```

- takes a 2-d array name and a 1-d array name
  - the `int` type 2-d arrays : (`X`)
  - the `double` type 1-d array : (`A`)
- returns nothing
- fill the `X[1]`, `X[2]`, and `X[3]` 1-d subarrays with random numbers
- `A[i]` is filled with the average value of `X[1][i]`, `X[2][i]`, and `X[3][i]`
- `X[0][i]` is filled with the expression `i+1+201600`
- all the elements of X are modified

### 1.3 pr_table() function

```
//--------------------------------------------------
// Print the original table
//-------------------------------------------------
void pr_table (int X[][SIZE], double A[])
{
  int i;

  printf("%10s %10s %10s %10s %10s \n", "StID",
         "Korean", "Enlgish", "Math", "Average");

  for (i=0; i<SIZE; ++i) {
    printf("%10d %10d %10d %10d %10.2f \n",
           X[0][i], X[1][i], X[2][i], X[3][i], A[i]);
    }
}
```

- takes a 2-d array name and a 1-d array name
  - the `int` type 2-d arrays : (`X`)
  - the `double` type 1-d array : (`A`)
- returns nothing
- print the student ID, test scores of Korean, English, and Mathematics, and their average score row by row
- header

| %10s | %10s | %10s | %10s | %10s |
|---|---|---|---|---|
| "StID" | "Korean" | "English" | "Math" | "Average" |

- each row

| %10d | %10d | %10d | %10d | %10.2f |
|---|---|---|---|---|
| X[0][i] | X[1][i] | X[2][i] | X[3][i] | A[i] |

- 
```
      StID     Korean    English       Math    Average
    201601         17         78         99      64.67
    201602         65         87         61      71.00
    201603         21         46         91      52.67
    201604        100          7         31      46.00
    201605         14         63         42      39.67
    201606         72         78         68      72.67
    201607          9        100         68      59.00
    201608         84         20          2      35.33
    201609         29         79         62      56.67
    201610         69         53         94      72.00
```

- no array is modified.

## 1.4 Dbubblesort() function

```
//-----------------------------------------------
// Bubble Sort Double Array
//----------------------------------------------
void DbubbleSort(double a[], int size)
{
  int p, j;
  double tmp;

  for (p=1; p< size; ++p) {
    for (j=0; j< size-1; ++j) {
      if ( a[j] < a[j+1] ) {
        tmp = a[j];
        a[j] = a[j+1];
        a[j+1] = tmp;
      }
    }
  }
}
```

- takes the 1-d array name of the `double` type and the array size
- returns nothing
- there are `size-1` passes : `for (p=1; p<size; ++p) { ... }`
- in each pass, perform the following basic operation over `size-1` pairs of adjacent elements : `for (j=0; j<size-1; ++j) { ... }`
  - for a given `j`, compare the pair `a[j]` and `a[j+1]`
  - if ( `a[j] < a[j+1]` ) then swap each other
  - thus ensuring `a[j]` is greater than or equal to `a[j+1]`
  - the next pair to be compared will be
    - `a[j+1]` and `a[j+2]` in terms of the old value of `j`
    - `a[j]` and `a[j+1]` in terms of the incremented value of `j`
- the current element is moved to the right (increasing index) until the smaller element is found
- after `p-1` passes, the array elements are in the increasing order.
- the given array is modified

## 1.5 pr_sorted_table() function

```
//--------------------------------------------------
// Print the Sorted Table
//-------------------------------------------------
void pr_sorted_table (int X[][SIZE], double A[])
{
  int i, j;
  double B[SIZE]; // Backup Array for Sorting

  for (i=0; i<SIZE; ++i) B[i] = A[i];

  //....................
  DbubbleSort(B, SIZE);
  //....................

  printf("\n\nSorted on a student's average\n\n");
  printf("%10s %10s %10s %10s %10s \n", "StID",
          "Korean", "Enlgish", "Math", "Average");

  for (i=0; i<SIZE; ++i) {
    for (j=0; j<SIZE; ++j) if (B[i] == A[j]) break;
    printf("%10d %10d %10d %10d %10.2f \n",
           X[0][j], X[1][j], X[2][j], X[3][j], A[j]);
  }
}
```

- takes a 2-d array name and a 1-d array name
  - the `int` type 2-d arrays : (`X`)
  - the `double` type 1-d array : (`A`)
- returns nothing
- initially, all the rows of `X` array and `A` array are sorted by the student ID
- copy `A` array into `B` array
- sort `B` array : `DbubbleSort(B, SIZE);`
- only `B` array are in the increasing order of the average score
- for each `B[i]` : `for (i=0; i<SIZE; ++i)`
  - find the index j such that `A[j]=B[i]` : `for (j=0; j<SIZE; ++j)`
  - print the (j+1)-th row of the original table
  -

| %10d | %10d | %10d | %10d | %10.2f |
|---|---|---|---|---|
| X[0][j] | X[1][j] | X[2][j] | X[3][j] | A[j] |

- no array is modified

## 1.6 Avg() function

```
//-----------------------------------------------------
// Average over Integer Array
//----------------------------------------------------
double Avg(int M[], int n) {
  int i; double S=0.0;

  for (i=0; i<n; ++i) S+= M[i];

  return S/n;
}
```

- takes a 1-d `int` array name and its size
- returns its arithmetic average value (a double type)
- $\frac{1}{n}\sum_{i=0}^{n-1} X[i]$

## 1.7 DAvg() function

```
//---------------------------------------------------
// Average over Doubl Array
//--------------------------------------------------
double DAvg(double N[], int n) {
  int i; double S=0.0;

  for (i=0; i<n; ++i) S+= N[i];

  return S/n;
}
```

- takes a 1-d `double` array name and its size
- returns its arithmetic average value (a double type)
- $\frac{1}{n} \sum_{i=0}^{n-1} Y[i]$

## 1.8 pr_averages() function

```
//---------------------------------------------------
// Print the Averages
//--------------------------------------------------
void pr_averages(int X[][SIZE], double A[]) {
  double A1 = Avg(X[1], SIZE);
  double A2 = Avg(X[2], SIZE);
  double A3 = Avg(X[3], SIZE);
  double A4 = DAvg(A, SIZE);

  printf("%10s %10.2f %10.2f %10.2f %10.2f \n",
        "Average", A1, A2, A3, A4);
}
```

- takes a 2-d array name and a 1-d array name
  - the `int` type 2-d arrays : (`X`)
  - the `double` type 1-d array : (`A`)
- returns nothing
- `A1` : the average score of the Korean subject
- `A2` : the average score of the English subject
- `A3` : the average score of the Mathematics subject
- `A4` : the average score of each student’s average score
-

| %10s | %10.2f | %10.2f | %10.2f | %10.2f |
|---|---|---|---|---|
| ”Average” | A1 | A2 | A3 | A4 |

- no array is modified

## 1.9 main() function

```
//======================================================
// main
//======================================================
int main(void) {
  // X[0][SIZE] --> I[SIZE]; // ID of a student
  // X[1][SIZE] --> K[SIZE]; // Grade of Korean
  // X[2][SIZE] --> E[SIZE]; // Grade of English
  // X[3][SIZE] --> M[SIZE]; // Grade of Math
  int X[4][SIZE];
  double A[SIZE]; // Average of a student

  init_arrays(X, A);
  pr_table(X, A);
  pr_sorted_table(X, A);
  pr_averages(X, A);
}
```

- declare a 2-d array and a 1-d array
  - `int X[4][SIZE];`
  - `double A[SIZE];`
- call `init_arrays()` function
- call `pr_table()` function
- call `pr_sorted_table()` function
- call `pr_averages()` function

## 1.10 When students have the same average

**test cases** the previous codes do not work in such case.

```
/--------------------------------------------------
// Initialize K[], E[], M[] arrays
// by assigning random number grade
//---------------------------------------------------
void init_arrays
(int I[], int K[], int E[], int M[], double A[])
{
  int i;

  // srand(7) makes rand() generate
  // the same random sequence
  // --> easy to debug a program
  srand(7);

  for (i=0; i<SIZE; ++i) {
    I[i] = i+1 + 201600;
    K[i] = rand() % 101;
    E[i] = rand() % 101;
    M[i] = rand() % 101;
    A[i] = avg3(K[i], E[i], M[i]);
  }

    X[1][3] = X[1][2];
    X[2][3] = X[2][2];
    X[3][3] = X[3][2];
       A[3] =    A[2];

    X[1][5] = X[1][2];
    X[2][5] = X[2][2];
    X[3][5] = X[3][2];
       A[5] =    A[2];

    X[1][7] = X[1][2];
    X[2][7] = X[2][2];
    X[3][7] = X[3][2];
       A[7] =    A[2];


}
```

- rows having index values of 2, 3, 5, 7 have the same score.
- only one student ID (201603) is repeated.

```
   StID     Korean    English       Math    Average
 201610         69         53         94      72.00
 201602         65         87         61      71.00
 201601         17         78         99      64.67
 201607          9        100         68      59.00
 201609         29         79         62      56.67
 201603         21         46         91      52.67
 201603         21         46         91      52.67
 201603         21         46         91      52.67
 201603         21         46         91      52.67
 201605         14         63         42      39.67
Average      28.70      64.40      79.00      57.37
```

## 1.11 Handling the same average cases - Method 1

**using extra memory**

- using index array : `int C[SIZE];`
- swap both average score and its index : `DbubbleSort()`

```
//-----------------------------------------------------
// Bubble Sort Double Array
//----------------------------------------------------
void DbubbleSort(double b[], int c[], int size)
{
  int p, j, t;
  double tmp;

  for (p=1; p< size; ++p) {
    for (j=0; j< size-1; ++j) {

      if ( b[j] < b[j+1] ) {
        tmp = b[j];
        b[j] = b[j+1];
        b[j+1] = tmp;

        t = c[j];
        c[j] = c[j+1];
        c[j+1] = t;
      }

    }
  }
}
```

```
//--------------------------------------------------
// Print the Sorted Table
//-------------------------------------------------
void pr_sorted_table (int X[][SIZE], double A[])
{
  int i, j;
  double B[SIZE]; // Backup Array for Sorting
  int    C[SIZE];

  for (i=0; i<SIZE; ++i) {
    B[i] = A[i];
    C[i] = i;
  }

  //....................
  DbubbleSort(B, C, SIZE);
  //....................

  printf("\n\nSorted on a student's average\n\n");
  printf("%10s %10s %10s %10s %10s \n", "StID",
          "Korean", "Enlgish", "Math", "Average");

  for (i=0; i<SIZE; ++i) {
    j = C[i];

    printf("%10d %10d %10d %10d %10.2f \n",
           X[0][j], X[1][j], X[2][j], X[3][j], A[j]);
  }
}
```

## 1.12 Handling the same average cases - Method 2

### using extra computation

- counting the same average cases: `int cnt;`
- search all indices for the same average : `j`

```
//---------------------------------------------------
// Print the Sorted Table
//--------------------------------------------------
void pr_sorted_table (int X[][SIZE], double A[])
{
  int i, j, cnt;
  double B[SIZE]; // Backup Array for Sorting

  for (i=0; i<SIZE; ++i) B[i] = A[i];

  //....................
  DbubbleSort(B, SIZE);
  //...................

  printf("\n\nSorted on a student's average\n\n");
  printf("%10s %10s %10s %10s %10s \n", "StID",
          "Korean", "Enlgish", "Math", "Average");

  for (i=0; i<SIZE; ++i) {
    cnt=1;
    while (B[i-cnt]==B[i]) cnt++;

    j=0;
    while (cnt>0) {
      cnt--;
      while (A[j]!=B[i]) j++;
      if (cnt > 0) j++;
    }

    printf("%10d %10d %10d %10d %10.2f \n",
           X[0][j], X[1][j], X[2][j], X[3][j], A[j]);
  }
}
```

# References

[1] C Programming Exercieses, https://cprogramex.wordpress.com/