Example 1 : using 1-d arrays

Using 1-d arrays

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10
```

avg3() definition

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

init_arrays() definition

```
//-------------------------------------------------
// Initialize K[], E[], M[] arrays
// by assigning random number scores
//-------------------------------------------------
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]);
   }
}
```

pr_table() definition

```
//-------------------------------------------------
// Print the original table
//------------------------------------------------
void pr_table
(int I[], int K[], int E[], int M[], double A[])
{
   int i;

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

   for (i=0; i<SIZE; ++i) {
     printf("%10d %10d %10d %10d %10.2f \n",
             I[i], K[i], E[i], M[i], A[i]);

   }
}
```

DbubbleSort() definition

```
//-----------------------------------------------------
// 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;
      }
    }
  }
}
```

pr_sorted_table() definition

```
//------------------------------------------------
// Print the Sorted Table
//------------------------------------------------
void pr_sorted_table
(int I[], int K[], int E[], int M[], 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", "English", "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",
             I[j], K[j], E[j], M[j], A[j]);
   }
}
```

Avg() definition

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

  for (i=0; i<n; ++i) S+= X[i];
  return S/n;
}
```

DAvg() definition

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

  for (i=0; i<n; ++i) S+= Y[i];
  return S/n;
}
```

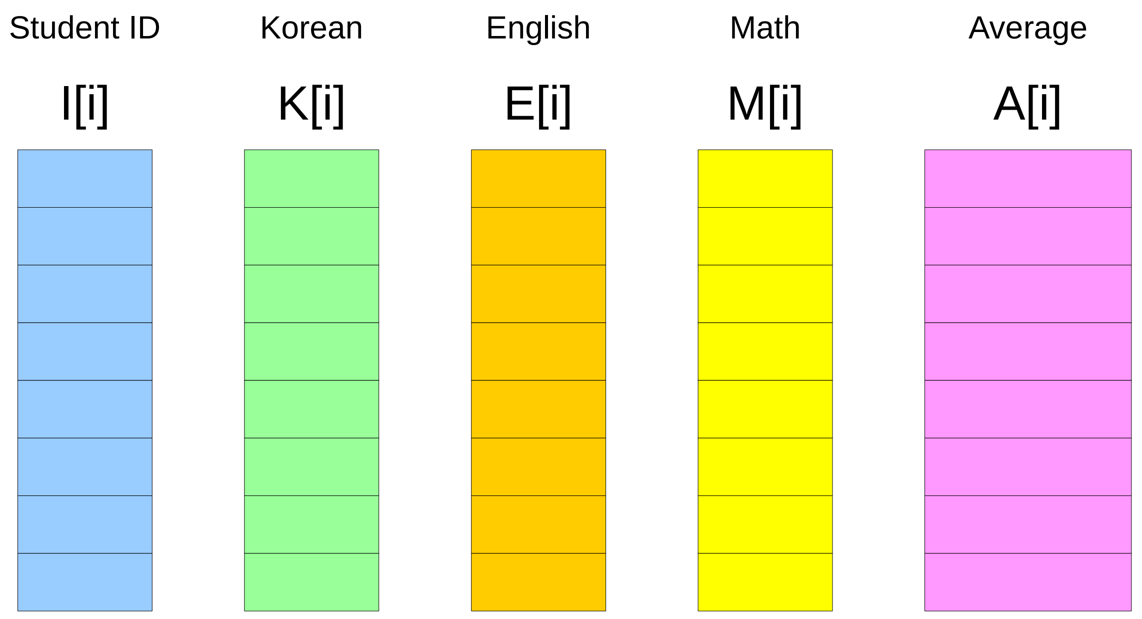
pr_averages() definition

```
//-------------------------------------------------------
// Print the Averages
//------------------------------------------------------
void pr_averages(int K[], int E[], int M[], double A[])
{
  double A1 =  Avg(K, SIZE);
  double A2 =  Avg(E, SIZE);
  double A3 =  Avg(M, SIZE);
  double A4 = DAvg(A, SIZE);

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

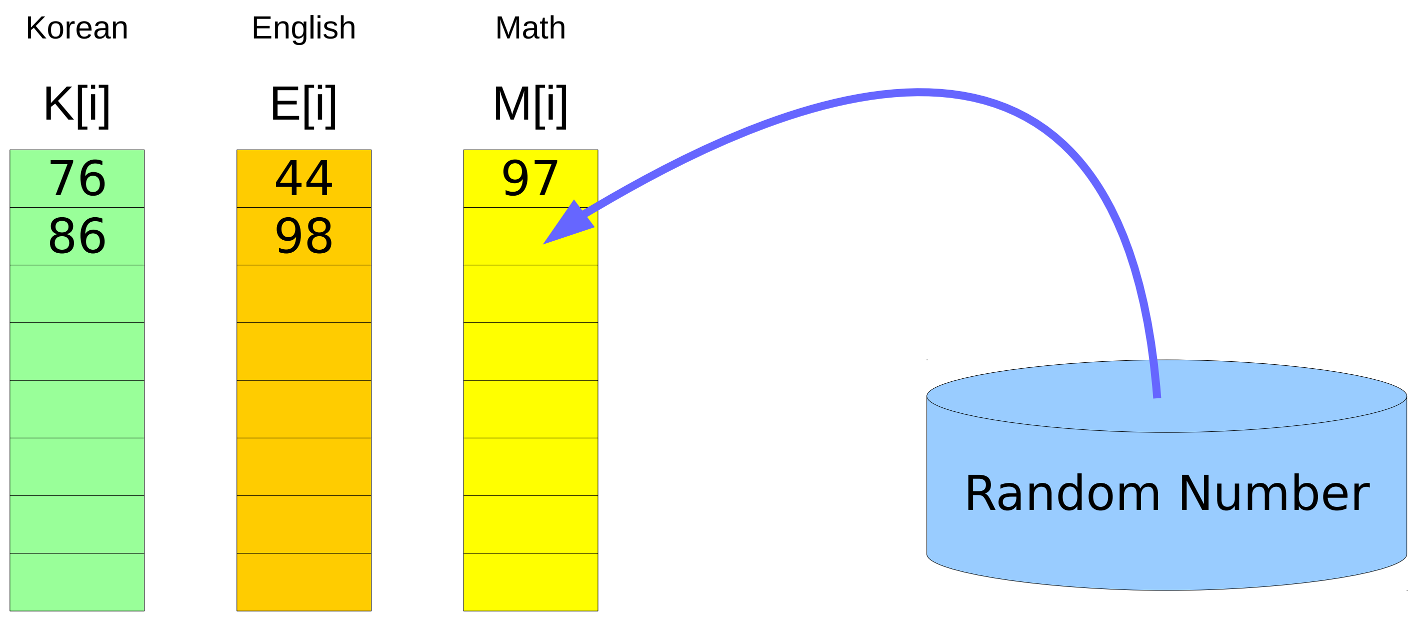
main() definition

```
int main(void) {
  int    I[SIZE];              // ID of a student
  int    K[SIZE];              // Korean subject score
  int    E[SIZE];              // English subject score
  int    M[SIZE];              // Math subject score
  double A[SIZE];              // Average score

  init_arrays(I, K, E, M, A);

  pr_table(I, K, E, M, A);

  pr_sorted_table(I, K, E, M, A);

  pr_averages(K, E, M, A);
}
```

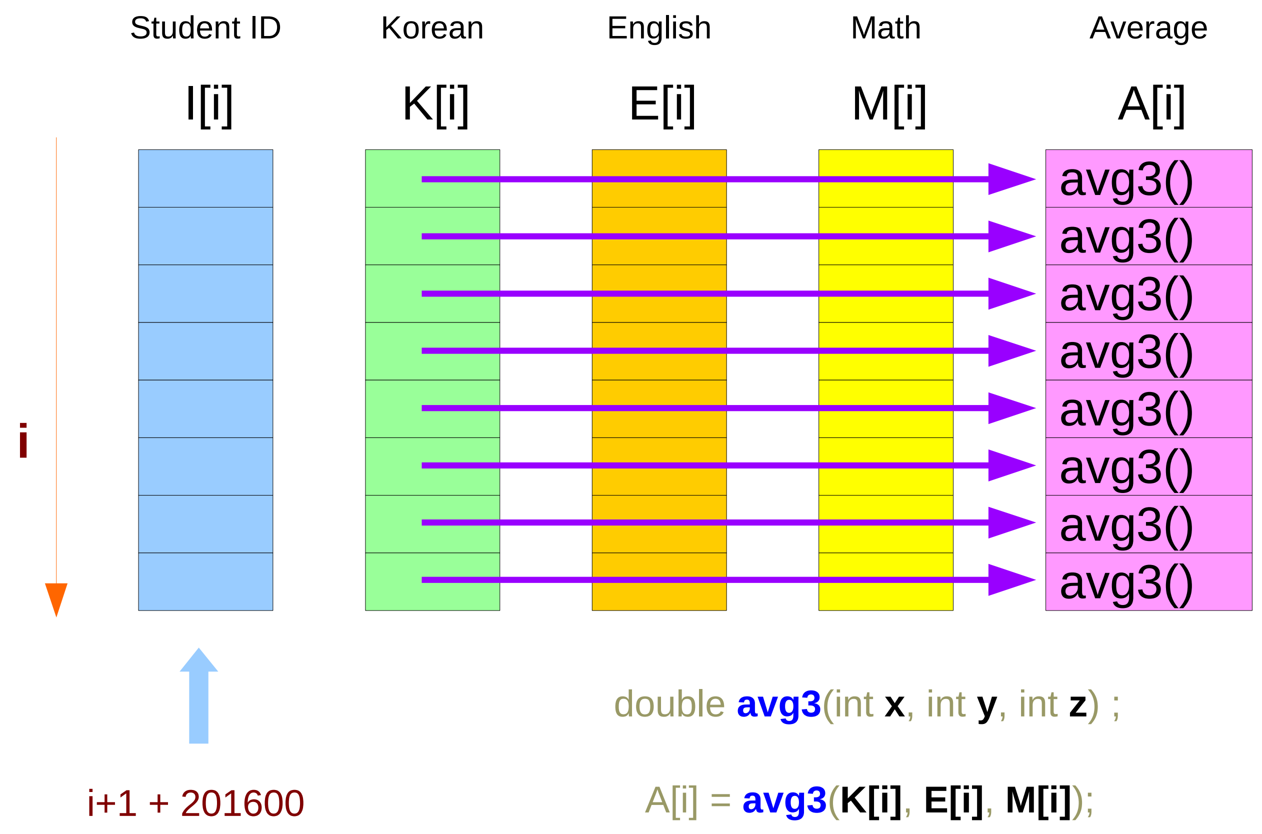
Using 1-d Arrays



init_arrays() - filling scores

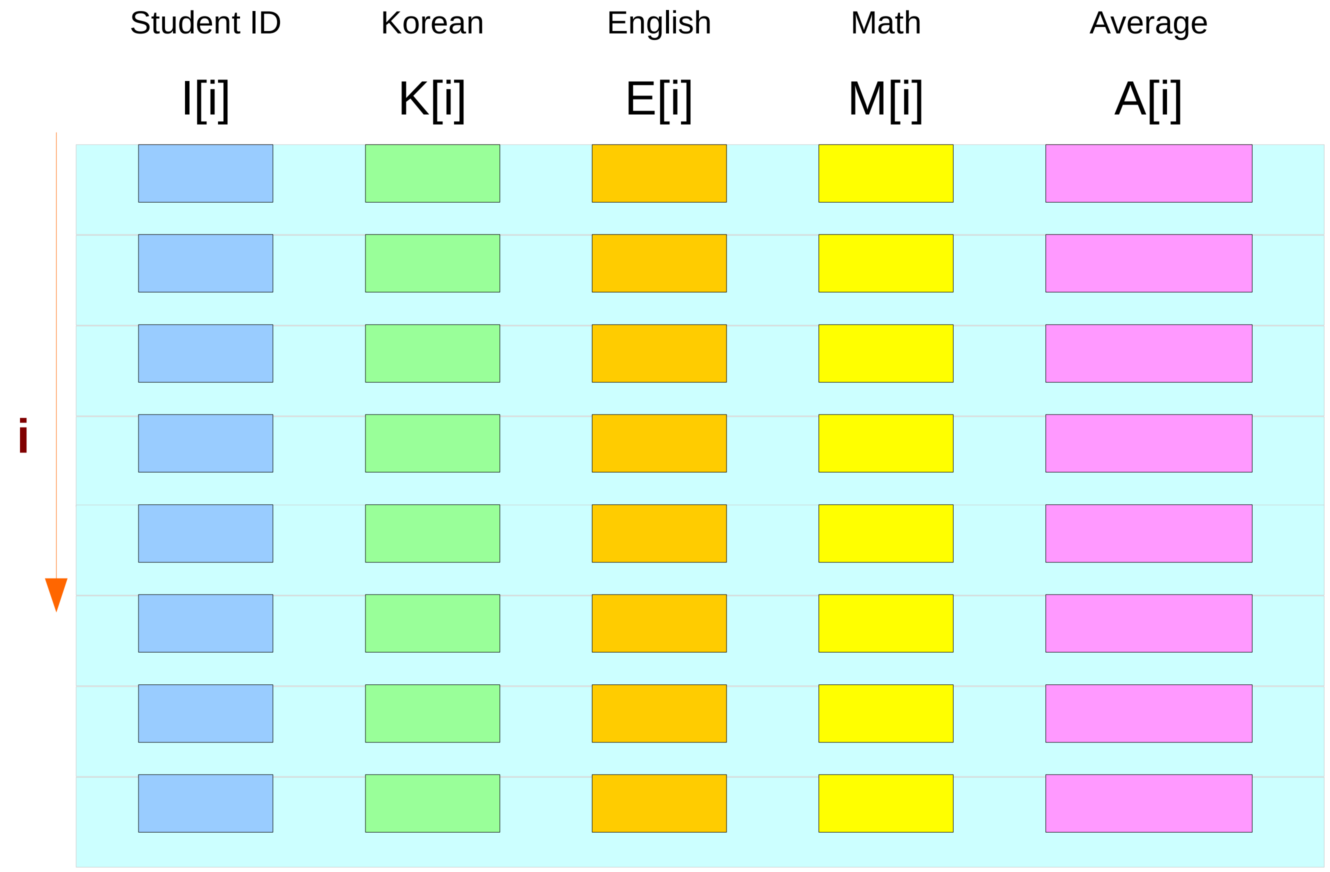


init_arrays() - computing averages

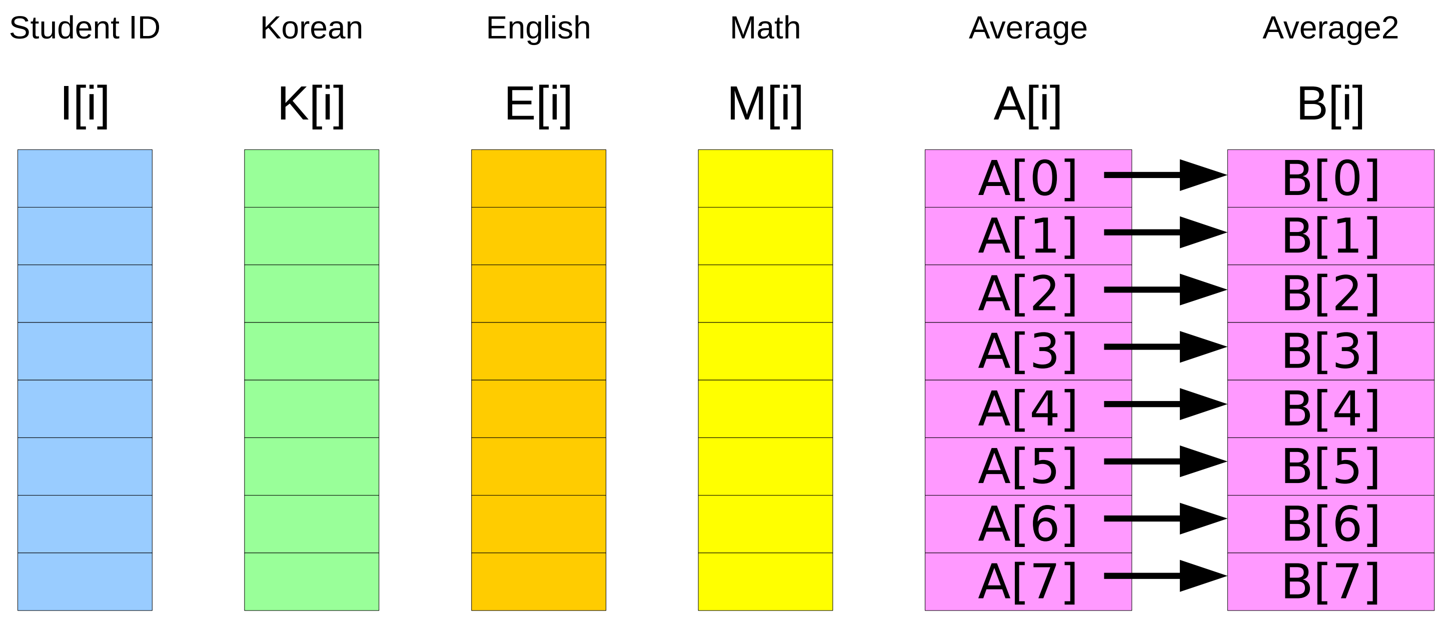


```
double avg3(int x, int y, int z) ;

A[i] = avg3(K[i], E[i], M[i]);
```

pr_table()



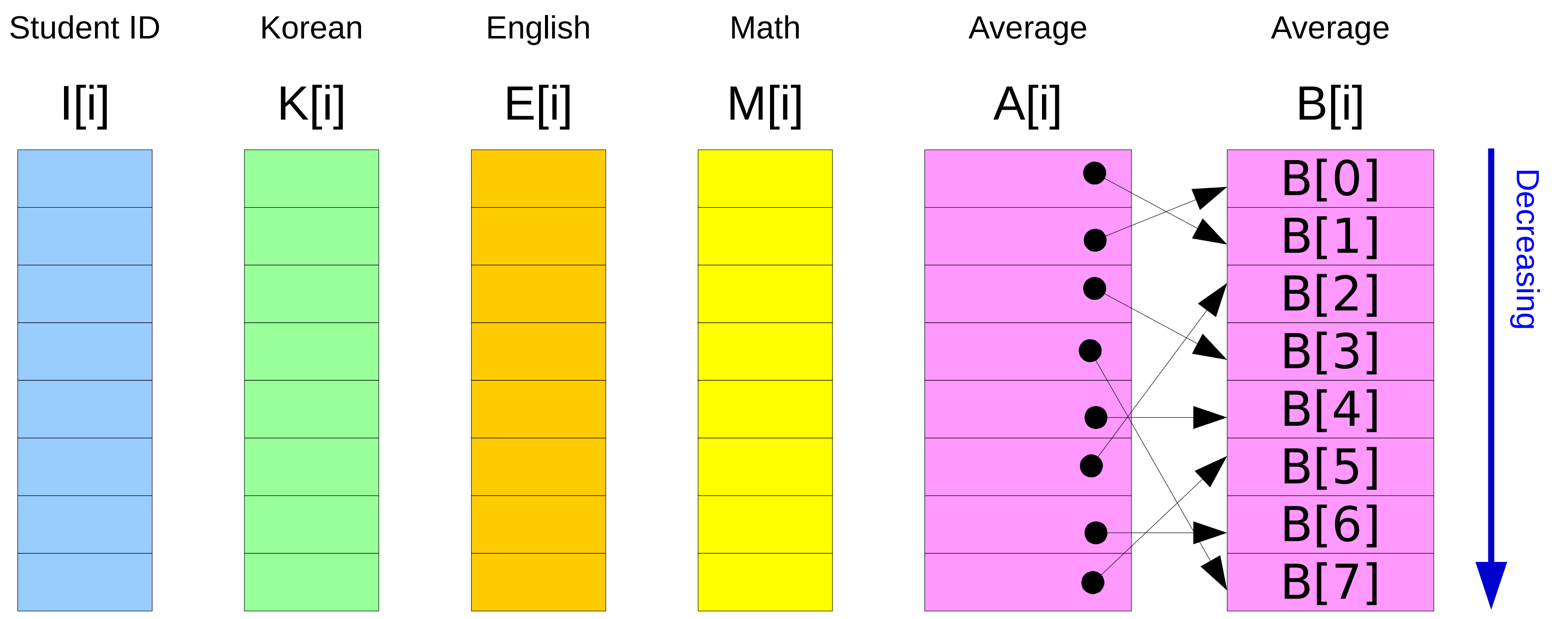
pr_sorted_table – copying A to B



First, copy
A[i] into B[i]

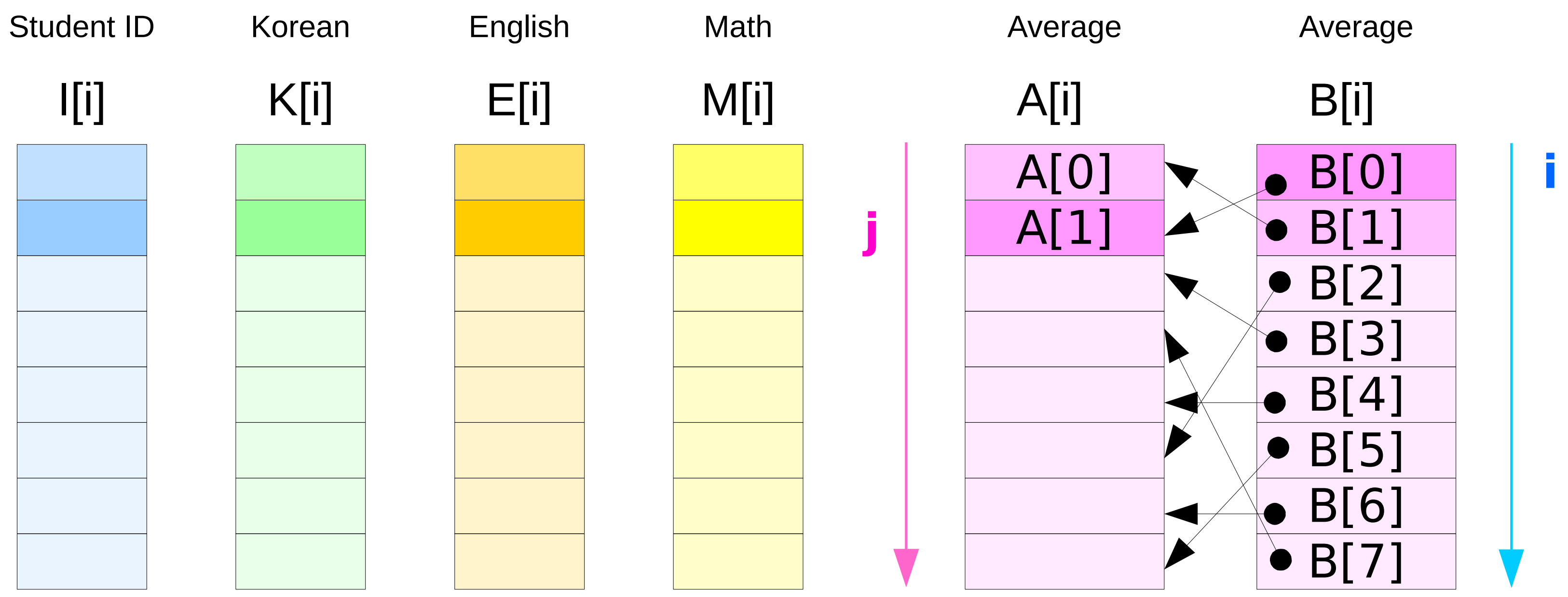
Assume that two averages have
always different values

pr_sorted_table – sorting B



after DbubbleSort()
B[i] > B[i]
A, B: different order

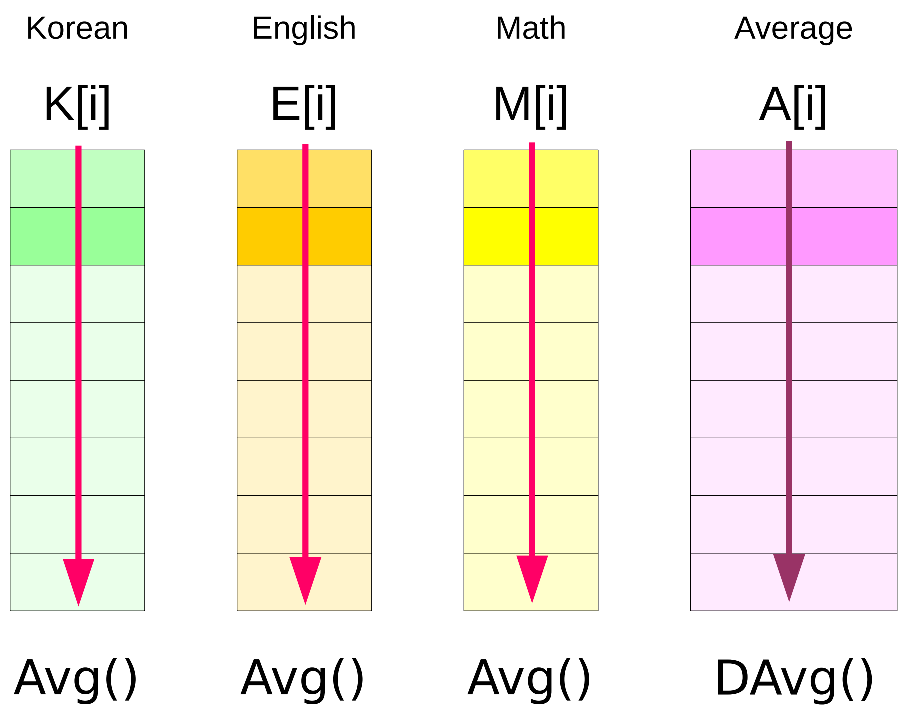
pr_sorted_table – printing by B



Search A[j] = B[i]

Assume that two averages have
always different values

pr_averages



```
double Avg(int X[], int n);

double DAvg(double Y[], int n);

double A1 =  Avg(K, SIZE);
double A2 =  Avg(E, SIZE);
double A3 =  Avg(M, SIZE);

double A4 = DAvg(A, SIZE);
```

Function Prototypes and Function Calls

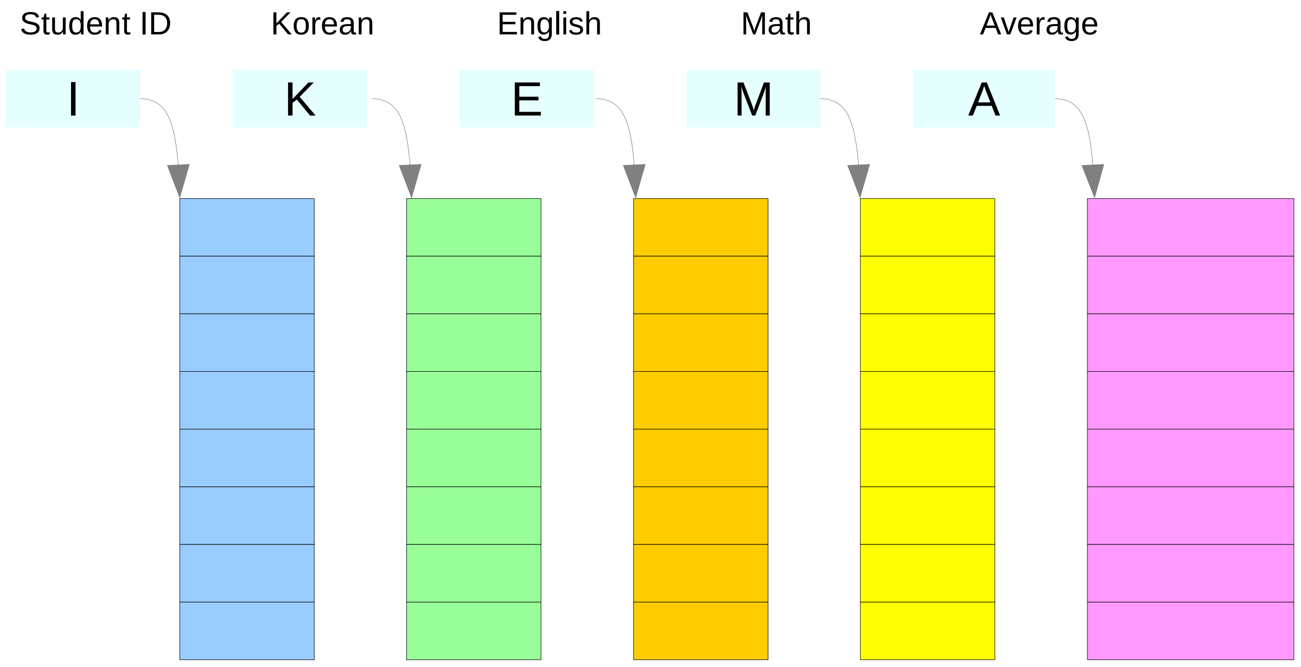
```
double   avg3              (int x, int y, int z) ;
void     init_arrays       (int I[], int K[], int E[], int M[], double A[]);
void     pr_table          (int I[], int K[], int E[], int M[], double A[]);
void     DbubbleSort       (double a[], int size);
void     pr_sorted_table   (int I[], int K[], int E[], int M[], double A[]);
double   Avg               (int X[], int n);
double   DAvg              (double Y[], int n);
void     pr_averages       (int K[], int E[], int M[], double A[]);
```

```
init_arrays(I, K, E, M, A); ………………………….  in main()
    A[i] = avg3(K[i], E[i], M[i]); ……………………  in init_arrays()

pr_table(I, K, E, M, A); …………………………….  in main()

pr_sorted_table(I, K, E, M, A); …………………..  in main()
    DbubbleSort(B, SIZE); ………………………  in pr_sorted_table()

pr_averages(K, E, M, A); ………………………….. in main()
    double A1 =  Avg(K, SIZE); …………………  in pr_averages()
    double A2 =  Avg(E, SIZE); …………………  in pr_averages()
    double A3 =  Avg(M, SIZE); …………………  in pr_averages()
    double A4 = DAvg(A, SIZE); …………………in pr_averages()
```

1-d Array Definitions

```
int main(void) {
  int       I[SIZE];      // ID of a student
  int       K[SIZE];      // Korean subject score
  int       E[SIZE];      // English subject score
  int       M[SIZE];      // Math subject score
  double A[SIZE];         // Average score

  init_arrays(I, K, E, M, A);

  pr_table(I, K, E, M, A);

  pr_sorted_table(I, K, E, M, A);

  pr_averages(K, E, M, A);
}
```

int K[SIZE] ... : Formal Parameter



int I[SIZE], K[SIZE], E[SIZE], M[SIZE]; double A[SIZE]; Array Name

I K E M A Starting Address

```
init_arrays      (int I[],  int K[],  int E[],  int M[],   double A[]);
pr_table         (int I[],  int K[],  int E[],  int M[],   double A[]);
pr_sorted_table  (int I[],  int K[],  int E[],  int M[],   double A[]);
pr_averages      (          int K[],  int E[],  int M[],   double A[]);
```

References

[1] Essential C, Nick Parlante
[2] Efficient C Programming, Mark A. Weiss
[3] C A Reference Manual, Samuel P. Harbison & Guy L. Steele Jr.
[4] C Language Express, I. K. Chun
[5] cprogramex.wordpress.com