5. Functions 2

Lecturer : Watcharin Sarachai, Ph.D.

Email : watcharin_s@mju.ac.th

Facebook : wsarachai

Topic

- Array
- Arrays and for loops
- Selection Sort

Call by Value vs. Call by Reference

- **Call by Value**:
  - เมื่อผ่านค่าตัวแปรชนิดพื้นฐานไปยังฟังก์ชัน จะเป็นการผ่านค่าสำเนาข้อมูลของตัวแปร
  - การเปลี่ยนแปลงใด ๆ ที่เกิดขึ้นกับค่าสำเนาตัวแปรจะไม่มีผลต่อค่าตัวแปรเดิมแต่อย่างใด
- **Call by Reference**:
  - เมื่อผ่านค่าตัวแปรอะเรย์หรือ Pointer ไปยังฟังก์ชัน จะเป็นการผ่านค่า reference ของตัวแปร
  - การเปลี่ยนแปลงใด ๆ ที่เกิดขึ้นกับค่า reference ตัวแปรจะมีผลกระทบโดยตรงกับค่าตัวแปรเดิมเสมอ

Passing Arrays to Functions

- การผ่านค่า *value* ในการส่งผ่านพารามิเตอร์ไปยังเมธอด ซึ่งการผ่านค่า ตัวแปรที่เป็นชนิดข้อมูลแบบพื้นฐาน จะมีความแตกต่างไปจากการผ่านค่าของอะเรย์
- การผ่านค่า ตัวแปรที่เป็นชนิดข้อมูลแบบพื้นฐานจะเป็นการผ่านค่าที่แท้จริงของ value ดังนั้นการเปลี่ยนแปลงใด ๆ ภายในเมธอดที่เป็น local จะไม่ส่งผลกระทบไปยัง value ของตัวแปรที่อยู่นอกเหนือเมธอด
- การผ่านค่าอะเรย์ไปยังเมธอดจะอยู่ในรูปของ Pointer ไปยังอะเรย์ ดังนั้นการเปลี่ยนแปลงใด ๆ ที่เกิดขึ้นภายในเมธอดจะก่อให้เกิดผลกระทบต่อตัวอะเรย์โดยตรงไม่ว่าจะอยู่ส่วนใดภายในโปรแกรม

Passing Arrays to Functions

- โดยปกติแล้วการผ่านค่าตัวแปรชนิดพื้นฐานไปยังฟังก์ชัน สามารถทำได้โดยการผ่านค่าสำเนาข้อมูลจากฟังก์ชันที่เรียกใช้และรับด้วยฟังก์ชันที่ถูกเรียกใช้ตามลำดับ เช่น

```
#include <stdio.h>

void test (int);
main ()
{
  int x = 5;
        printf ("In main x:  %d\n", x);
        test (x);
        printf ("In main x:  %d\n", x);
}

void  test (int x) {
       x += 10;
       printf ("In Test x:  %d\n", x);
}
```

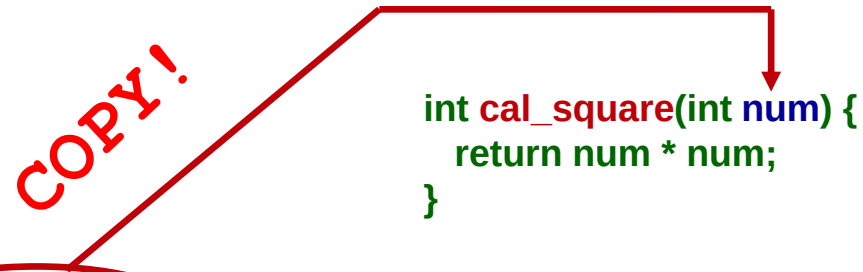
Output:

```
In main x:  5
In Test x:  15
In main x:  5
```

Array : Pass by Values

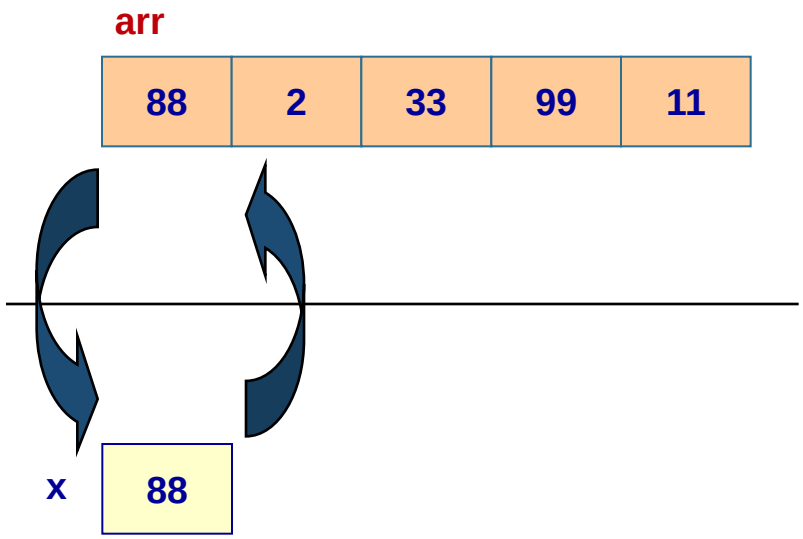
- แต่ละ element ของอะเรย์จะทำหน้าที่เช่นเดียวกับตัวแปรทั่ว ๆ ไป
- นั่นคือสามารถผ่านค่าไปยังฟังก์ชันได้ทีละค่าในรูปของ arguments

```
int calc_square(int num);

main () {
   int numbers[5];
   int i, square;
   for (i=0; i<5; i++)
   {
         numbers[i] = i * 10;
         square = cal_square( numbers[i] );
         printf("%d squared is %d\n", numbers[i], square);
   }
}
```



Array : Pass by Values

```
#include <stdio.h>
int     change_value  (int);
main()                                  //64
{
   int arr[ ] = { 88, 2, 33, 99,
     11 };
   int i;
     for ( i = 0; i < 5; i++ ) {
       arr[i] = change_value(arr[i]);
              printf(" %d ", arr[i]);
     }
}
int     change_value ( int x )
{
    return (x +2);
}
```



Output: 90 4 35 101 13

Array : Pass by Reference

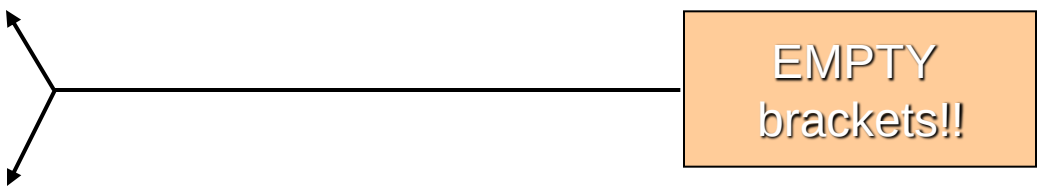
- การผ่านค่าไปยังฟังก์ชันโดยใช้อะเรย์สามารถทำได้โดยการกำหนดPrototypes ได้ดังต่อไปนี้:
  - การใช้พารามิเตอร์ในรูปของอะเรย์จะต้องระบุ brackets [ ] ด้วยเสมอ
  - ไม่จำเป็นต้องระบุขนาดไว้ใน brackets

```
#include <stdio.h>
void funcName (int []);

main () {
    int temp[4] = {75, 65, 89, 72};
            funcName(temp);
}

void funcName (int tempArray[]) {

}
```

Arrays & Functions

- การผ่านค่าอะเรย์ในรูปของอาร์กิวเมนต์ไปยังฟังก์ชันมีขั้นตอนดังนี้ ?
- STEP 1: ประกาศฟังก์ชันที่ต้องการ :

```
void printArray( int arr[ ], int size); /*prototype*/
. . .
. . .

/* definition */
void printArray( int arr[ ], int size) {
int i;
        for(i=0; i<size; i++)
              printf("%d\n", arr[i] );
}
```



Arrays & Functions

- STEP 2: เรียกใช้ฟังก์ชันโดยการระบุอาร์กิวเมนต์ที่เป็นชื่อของอะเรย์

```
main()  {
      int numbers[SIZE];
      int i;
      for(i=0; i<SIZE; i++)
                  numbers[i] = i+10;
      printArray(numbers, SIZE);
}
```

Array name ONLY--No brackets at all!

Arrays & Functions

- การผ่านค่าอะเรย์ไปยังฟังก์ชันสามารถกำหนดผ่าน prototype ได้ดังต่อไปนี้ :

```
void printArray( int arr[ ], int size);
```

- Recall: การประกาศฟังก์ชันจะมีผลทำให้ตัวแปรจาก argument list กลายเป็นแบบ Local
- Recall: ฟังก์ชันที่ถูกเรียกใช้จะสำเนาค่า argument เข้าสู่ parameter ที่กำหนดไว้
- ดังนั้น:
  - int size หมายถึงฟังก์ชัน printArray ประกอบด้วยตัวแปร int ที่ชื่อ size
  - int arr[ ] หมายถึงฟังก์ชัน printArray มีตัวแปรอะเรย์ที่ชื่อ arr
- แต่ชื่อของอะเรย์ที่ใช้เป็น arguments จะแตกต่างไปจากชื่อตัวแปรชนิดอื่น ๆ
  - ฟังก์ชันจะสำเนา ADDRESS ของอะเรย์ ไม่ใช่ค่าที่อยู่ภายในอะเรย์

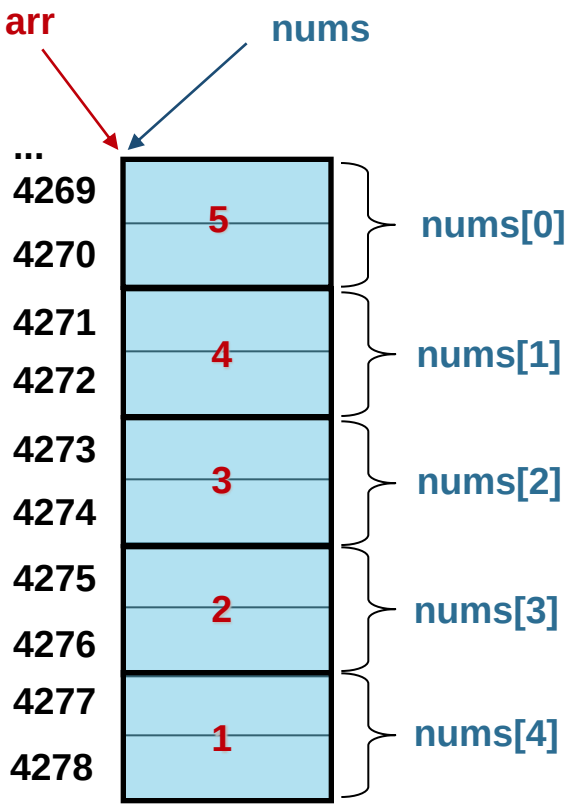
Arrays & Functions

- ชื่อของอะเรย์เป็นการนำเสนอแอดเดรสเริ่มต้นของอะเรย์
- ชื่อของอะเรย์ที่เป็น arguments จะไม่มีการสำเนาข้อมูลทั้งหมดของอะเรย์ แต่จะสำเนาเฉพาะค่าแอดเดรสเริ่มต้นของอะเรย์นั้น ๆ
  - ซึ่งในกรณีนี้จะเป็นการบอกฟังก์ชันถึงตำแหน่งที่ต้องการหาค่าของอะเรย์
- Effect: การทำงานในลักษณะนี้จะยอมให้ฟังก์ชันสามารถเปลี่ยนแปลงค่าของอะเรย์ที่ถูกรับเข้ามาในรูปของ "input" argument ได้

Arrays & Functions

```
void cleanArray( int [ ], int );
void printArray( int [ ], int );
main()  {
    int nums[5] = {5,4,3,2,1};

    cleanArray(nums,5);
    printArray(nums, 5);

}

void cleanArray( int arr[ ] , int size) {
    int i;
    for (i=0; i<size; i++)
          arr[i] =0;
}

 void printArray( int arr[ ], int size) {
     int i;
       for(i=0; i<size; i++)
             printf("%d\n", arr[i] );
}
```

//65



Arrays & Functions

```
void cleanArray ( int arr[ ], int size);

main()  {
   int nums[5] = {5,4,3,2,1};

   cleanArray( nums, 5);
   printArray( nums, 5);

}
```

- ฟังก์ชัน cleanArray() สำเนาแอดเดรสไปยังตัวแปรอะเรย์ arr
- นั่นหมายความว่า arr เริ่มต้นที่เมมโมรี่ตำแหน่งเดียวกันกับตัวแปรอะเรย์ nums ดังนั้นการเปลี่ยนแปลงใด ๆ ที่เกิดขึ้นกับ arr ย่อมส่งผลกระทบโดยตรงต่อ nums
- เมื่อสิ้นสุดการทำงานของฟังก์ชัน cleanArray() ค่าสมาชิกภายใน num จะมีค่าเป็น 0 ทั้งหมด

Call by Reference

- จากตัวอย่างโปรแกรมจะเป็นการส่งผ่านค่าแบบ: Call by Reference:
  - การผ่านค่าใช้ชื่อของอะเรย์ที่ถือเป็นแอดเดรสเริ่มต้นของอะเรย์
  - ดังนั้นจึงถือเป็นการผ่านค่า reference ที่อ้างอิงไปยังค่าตัวแปรอะเรย์เดิม
  - การเปลี่ยนแปลงที่เกิดขึ้นกับ reference จะมีผลต่อค่าอะเรย์เดิมเสมอ
- การผ่านค่าในอะเรย์จะดำเนินการแบบ Call by Reference เสมอ

Call by Reference

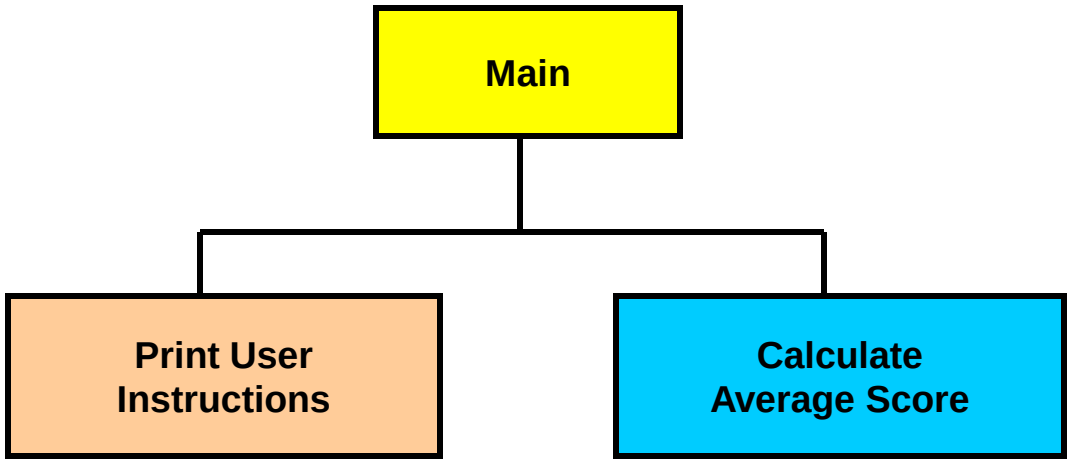
```
#include <stdio.h>
void  find_max( int [ ] );                          //66
main()  {
      int p[ ] = { 88, 2, 33, 99, 11 };
      int i;
            printf("Address of array in main = %p\n", &p[0]);
            for ( i = 0; i < 5; i++ )
                printf("%d", p[i]);
            find_max( p);
}
void  find_max( int x[] ) {
int  i = 0,  max = 0;
      printf("\nAddress of array in function = %p\n", &x[0]);
            for ( i = 0; i < 5; i++ )   {
                if ( max < x[i])
                   max = x[i];
              }
       printf( ."Maximum number is %d\n", max );
}
```

Output :

```
Address of array in main = FFFC
88   2   33   99   11
Address of array in function = FFFC
Maximum number is 101
```

Example Using Arrays

- Problem: หาค่าเฉลี่ย (average) ของคะแนนสอบและจำนวนของผู้ได้เกรด A's, B's, C's, D's, และ F's ตามลำดับ
- Design:



“Clean” Example Using Arrays

```
#include <stdio.h>

#define  SIZE 35                        /* number of tests                          */
#define  GRADES 5                       /* number of different grades: A, B, C, D, F */

void printInstructions ( ) ;
double findAverage (double sum, int quantity) ;

main ( )
{
   int i ;                              /* loop counter                             */
   int total ;                          /* total of all scores                      */
   int score [SIZE] ;                   /* student scores                           */
   int gradeCount [GRADES] ;            /* count of A’s, B’s, C’s, D’s, F’s */
   double average ;                     /* average score                            */

   /* Print the instructions for the user */

   printInstructions ( ) ;
```

“Clean” Example Using Arrays

```
/* Initialize grade counts to zero */

for ( i = 0; i < GRADES; i++ )
{
   gradeCount [ i ] = 0 ;
}

/* Fill score array with scores */

for ( i = 0; i < SIZE; i++ )
{
   printf (“Enter next score: ”) ;
   scanf (“%d “, &score [ i ] ) ;
   while ( score [ i ]< 0 || (score [ i ]>100 )
   {
      printf (“Scores must be between”) ;
      printf (“ %d and %d\n”, 0, 100) ;
      printf (“Enter next score : ”) ;
      scanf (“%d “, &score [ i ] ) ;
   }

}
```

```
/* Calculate score total and count
   number of each grade */

for ( i = 0; i < SIZE; i++ )
{
   total += score [ i ] ;

   switch ( score [ i ] / 10 )
   {
      case 10 :
      case   9 :  gradeCount [4]++;    break ;
      case   8 :  gradeCount [3]++;    break ;
      case   7 :  gradeCount [2]++;    break ;
      case   6 :  gradeCount [1]++;    break ;
      default  :  gradeCount [0]++;
   }
}
```

“Clean” Example Using Arrays

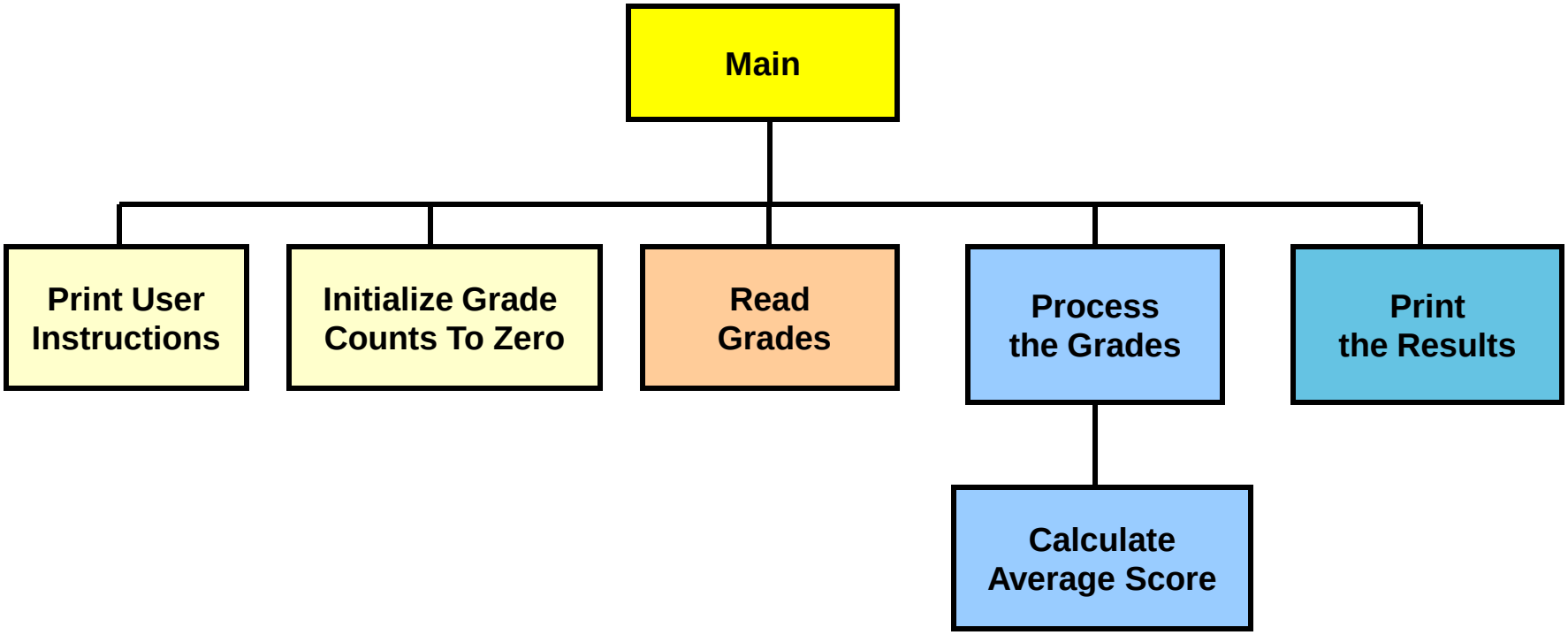
```
        /* Calculate the average score */
        average = findAverage (total, SIZE) ;

       printf (“The class average is %.2f\n”, average ) ;
       printf (“There were %2d As\n”, gradeCount [4] ) ;
       printf (“                %2d Bs\n”, gradeCount [3] ) ;
       printf (“                %2d Cs\n”, gradeCount [2] ) ;
       printf (“                %2d Ds\n”, gradeCount [1] ) ;
       printf (“                %2d Fs\n”, gradeCount [0] ) ;
    }

/*****************************************************************
** printInstructions - prints the user instructions
** Inputs:  None
** Outputs:  None
*****************************************************************/
void printInstructions ( )
{
   printf (“This program calculates the average score\n”) ;
   printf (“for a class of 39 students.  It also reports the\n”) ;
   printf (“number of A’s, B’s, C’s, D’s, and F’s.  You will\n”) ;
   printf (“be asked to enter the individual scores.\n”) ;
}
```

“Clean” Example Using Arrays

```
/*************************************************************
** findAverage - calculates an average
** Inputs:  sum - the sum of all values
**              num - the number of values
** Outputs:  the computed average
*************************************************************/

double findAverage (double sum, int num)
{
   double average ;   /* computed average */

    if ( num != 0 ) {
        average = sum / num ;
    }
    else {
        average = 0 ;
    }

   return average ;
}
```

Grades Program Using Pass by Reference

- ออกแบบเพิ่มเติมเพื่อให้สามารถบำรุงรักษาได้ง่ายขึ้น



“Clean” Grades Program (cont.)

```
#include <stdio.h>

#define  SIZE      35
#define  GRADES     5
#define  A          4
#define  B          3
#define  C          2
#define  D          1
#define  F          0
#define  MAX      100
#define  MIN        0

void    printInstructions ( ) ;
void    initArray (int anArray[ ], int size) ;
void    fillArray (int anArray[ ], int size) ;

double processGrades (int score[ ], int size, int gradeCount[ ] ) ;
double findAverage (double sum, int num) ;
void    printResults (double average, int gradeCount[ ] ) ;
```

“Clean” Grades Program (cont.)

```
main ( )
{
   int score [SIZE];                    /* student scores                  */
   int gradeCount [GRADES] ;            /* count of A's, B's, C's, D's, F's */
   double average;                       /* average score                  */
        printInstructions ( ) ;
        initArray (gradeCount, GRADES) ;
        fillArray (score, SIZE) ;
        average = processGrades (score, SIZE, gradeCount ) ;
        printResults (average, gradeCount) ;
}
 /*****************************************************************
 ** printInstructions - prints the user instructions
 /*****************************************************************
 void printInstructions ( )
 {
    printf ("This program calculates the average score\n") ;
    printf ("for a class of 39 students.  It also reports the\n") ;
    printf ("number of A's, B's, C's, D's, and F's.  You will\n") ;
    printf ("be asked to enter the individual scores.\n") ;
 }
```

“Clean” Grades Program (cont.)

```
/*****************************************************************
/* initArray - initializes an integer array to all zeros
/*****************************************************************/
void initArray (int anArray [ ], int size)
{
   for ( i = 0; i < size; i++ )    {
      anArray [ i ] = 0 ;
   }
}

 /*************************************************************
 ** findAverage - calculates an average
 *************************************************************/
 double findAverage (double sum, int num)     {
    double average ;   /* computed average */
     if ( num != 0 ) {
          average = sum / num ;
     }
     else {
          average = 0 ;
    }
    return average ;
 }
```

“Clean” Grades Program (cont.)

```
/*****************************************************************************
** fillArray - fills an integer array with valid values that are entered by the user.
**             Assures the values are between MIN and MAX.
** Inputs:  anArray - array to fill
** Outputs:  size - size of the array
** Side Effect - MIN and MAX must be #defined in this file
*****************************************************************************/

void fillArray (int anArray [ ], int size)
{
    int i ;
    for ( i = 0; i < size; i++ ) {
        printf (“Enter next value : ”) ;
        scanf (“%d “, &anArray [ i ] ) ;
        while ( (anArray [ i ] < MIN)  ||  (anArray [ i ] > MAX) ) {
            printf (“Values must be between %d and %d\n ”, MIN, MAX) ;
            printf (“Enter next value : ”) ;
            scanf (“%d “, &anArray[ i ] ) ;
        }
    }
}
```

“Clean” Grades Program (cont.)

```
/*************************************************************************
** processGrades - counts the number of A's, B's, C's, D's, and F's, and
**                 computes the average score
*************************************************************************/
double ProcessGrades (int score [ ], int size, int gradeCount [ ] )
{
   int total = 0;          /* total of all scores */
   double average;   /* average score      */

   for ( i = 0 ; i < size ; i++) {
      total += score [ i ] ;
      switch ( score [ i ] / 10 )       {
         case 10 :
         case   9 :  gradeCount [A]++  ;     break ;
         case   8 :  gradeCount [B]++ ;      break ;
         case   7 :  gradeCount [C]++ ;      break ;
         case   6 :  gradeCount [D]++        break ;
         case   5 :
         case   4 :
         case   3 :
         case   2 :
         case   1 :
         case   0 :gradeCount [F]++ ;        break ;
         default  : printf (“Error in score.\n”) ;
      }
   }

   average = findAverage (total, size) ;

   return average ;

}
```

“Clean” Grades Program (cont.)

```
/*****************************************************************************
** printResults - prints the class average and the grade distribution for
**                the class.
*****************************************************************************/

void  printResults (double average, int gradeCount [ ] )
{
   printf (“The class average is %.2f\n”, average ) ;
   printf (“There were %2d As\n”, gradeCount [A] ) ;
   printf (“             %2d Bs\n”, gradeCount [B] ) ;
   printf (“             %2d Cs\n”, gradeCount [C] ) ;
   printf (“             %2d Ds\n”, gradeCount [D] ) ;
   printf (“             %2d Fs\n”, gradeCount [F] ) ;
}
```

Question