```
/* copy one int array to another int array. Since this array is not a character array
(string), the function parameters must include the number of array elements. const
prevents change by assignment, incrementing, or decrementing.  c standard - If a
constant and/or volatile keyword is next to a type specifier, it applies to the type
specifier. Otherwise it applies to the pointer asterisk on its immediate left.
*/
#define SIZE 9
#include <stdio.h>
#include<conio.h>

void display_all(const int *iNum1_ptr,  int const *iNum2_ptr, int iSize);
void copy_all(const int iNum1_ptr[],  int iNum2_ptr[], int iSize);
/*-------------------------------------------------------------------------*/
int main(void)
{
  int iNum1_array[] = {1,3,5,7,9,11,13,15,17};
  int iNum2_array[SIZE];

  copy_all(iNum1_array, iNum2_array, SIZE);
  display_all(iNum1_array, iNum2_array, SIZE);

  return 0;
}/*-------------------------------------------------------------------------*/
void copy_all( const int iNum1_ptr[],  int iNum2_ptr[], int iSize)

             // (with array notation) the const type qualifier makes the iNum1_ptr
             // array elements read only and makes the pointer iNum1_ptr read only
{
  int iIndex;

  for (iIndex = 0; iIndex < iSize; iIndex++)
      iNum2_ptr[iIndex] = iNum1_ptr[iIndex];

//  *(iNum2_ptr + iIndex) = *(iNum1_ptr + iIndex);   ok
//  *(iNum2_ptr++) = *(iNum1_ptr + iIndex);          ok
//  *(iNum2_ptr++) = *(iNum1_ptr++);    error can't modify a constant object
//                                      iNum1_ptr is a constant
}/*-------------------------------------------------------------------------*/
void display_all(const int *iNum1_ptr, int const *iNum2_ptr, int iSize)

  // (with pointer notation) the const type qualifier makes what iNum1_ptr
            //  points to read only but does not make the pointer iNum1_ptr read only
{
  int iIndex;

      for (iIndex = 0; iIndex < iSize; iIndex++)
      printf("%5d ", *iNum1_ptr++);            // same as *(iNum1_ptr++)
      printf(" \n");

  for (iIndex = 0; iIndex < iSize; iIndex++)
    printf("%5d ",*(iNum2_ptr + iIndex));
    printf("\n\n");

//           *iNum1_ptr = 2;     error cannot modify a constant object
//  iNum1_ptr[1] = 88;  error cannot modify a constant object
//  iNum1_ptr++;        ok
}/*-------------------------------------------------------------------------*/
/*
    1     3     5     7     9    11    13    15    17
    1     3     5     7     9    11    13    15    17

Push a key to return to editor */
```

Commented [KW1]: The const qualifier is used to make sure the called function cannot change the actual parameter array contents.

Commented [KW2]: The array size should be passed to make sure no elements outside of the array boundary are accessed.

Commented [KW3]: iNum1_ptr[] is declaring an array, and at the same time is stating that an address, or pointer, must be passed as the actual parameter in this field.

Commented [KW4]: This shows two ways to declare an array. The first is initialized to the size of the elements in the {} braces and the second array is uninitialized to 9 elements.

Commented [KW5]: The name of an array is actually a constant address for it's element zero: &iNum1_array[0].

Commented [KW6]: A for loop is used to copy each element from iNum1_array into iNum2_array. Note the syntax used for iIndex.

Commented [KW7]: A for loop is used for printing the array elements using an incremented pointer syntax.

Commented [KW8]: A for loop is used for printing the array elements using pointer notation that adds the index to it's base value. Note the location of the dereferencing * and ().

```
/* both a static array and a global array are initialized to zero an automatic array
is uninitialized the scope of a global array is from definition to the end of the file
   pointer and array notation for a value and an address */
#define MAX 4        // number of array elements
#include<stdio.h>
#include<conio.h>

int iC_array[MAX];
void display_both(const int *iA_ptr, const int *iB_ptr);
/*---------------------------------------------------------------------*/
int main(void)
{
  static int iA_array[MAX];      // storage class static
  int iB_array[MAX];             // storage class automatic

  display_both(iA_array,iB_array);
  printf ("iB_array + 3 = %p   *(iB_array + 3) = %d \n", iB_array + 3, *(iB_array +
3));
  printf ("&iB_array[3] = %p   iB_array[3]     = %d \n",  &iB_array[3], iB_array[3]);

  return 0;
}
/*---------------------------------------------------------------------*/
void display_both(const int *iA_ptr, const int *iB_ptr)
{
      int iIndex;
       for (iIndex = 0; iIndex < MAX; iIndex++, iA_ptr++)
         printf("iA_ptr= %-4p          *iA_ptr= %-6d    &iA_ptr= %-6p\n",
               iA_ptr, *iA_ptr, &iA_ptr);
       printf("\n");

       for (iIndex = 0; iIndex < MAX; iIndex++, iB_ptr++)
         printf("iB_ptr= %-4p          *iB_ptr= %6d    &iB_ptr= %-6p\n",
               iB_ptr, *iB_ptr, &iB_ptr);
       printf("\n");

       for (iIndex = 0; iIndex < MAX; iIndex++)
         printf("&iC_array[%d]= %-4p    iC_aray[%d]= %-4d   iC_array= %-6p\n",
               iIndex,&iC_array[iIndex],iIndex, iC_array[iIndex], iC_array);
           printf("\n");
} /*---------------------------------------------------------------------*/
/*
iA_ptr= 012EA148          *iA_ptr= 0         &iA_ptr= 00D9FA34
iA_ptr= 012EA14C          *iA_ptr= 0         &iA_ptr= 00D9FA34
iA_ptr= 012EA150          *iA_ptr= 0         &iA_ptr= 00D9FA34
iA_ptr= 012EA154          *iA_ptr= 0         &iA_ptr= 00D9FA34

iB_ptr= 00D9FB0C          *iB_ptr= -858993460    &iB_ptr= 00D9FA38
iB_ptr= 00D9FB10          *iB_ptr= -858993460    &iB_ptr= 00D9FA38
iB_ptr= 00D9FB14          *iB_ptr= -858993460    &iB_ptr= 00D9FA38
iB_ptr= 00D9FB18          *iB_ptr= -858993460    &iB_ptr= 00D9FA38

&iC_array[0]= 012EA138    iC_aray[0]= 0      iC_array= 012EA138
&iC_array[1]= 012EA13C    iC_aray[1]= 0      iC_array= 012EA138
&iC_array[2]= 012EA140    iC_aray[2]= 0      iC_array= 012EA138
&iC_array[3]= 012EA144    iC_aray[3]= 0      iC_array= 012EA138

iB_array + 3 = 00D9FB18   *(iB_array + 3) = -858993460
&iB_array[3] = 00D9FB18   iB_array[3]     = -858993460
Press any key to continue . . . */
```

Commented [KW9]: A Global Array that every function has access to – this is not recommended. It is also initialized to all 0's by default.

Commented [KW10]: static initializes the array elements to all 0's and is alive for the entire program even when declared in user functions.

Commented [KW11]: Two ways to use/access the same array data -> pointer and array index.

Commented [KW12]: Left justify the data.

```
/* initialization of arrays pointers can be incremented, and can be dereferenced using
[] array names are constant and cannot be incremented a 4 element array used as a 5
element array, gives no compiler errors
*/
#include<stdio.h>
#include<conio.h>
#define SIZE 4
/*------------------------------------------------------------------*/
int main(void)
{
  int iArray1[SIZE], iArray2[SIZE], *iArray_ptr, iIndex;

  iArray_ptr = iArray1;
  iArray1[0] = 10;
  *(iArray1 + 1) = 11;
  *(iArray_ptr + 2) = 12;
  iArray_ptr[3] = 13;

  for (iArray_ptr = iArray1, iIndex = 0; iIndex < SIZE; iIndex++)
            printf("iArray1[%d] = %d   ",iIndex, *iArray_ptr++);
  printf("\n\n");

  for (iArray_ptr = iArray2, iIndex = 0;  iIndex < SIZE;  iIndex++)
            {
            printf("Enter an integer  ");
            scanf("%d" , iArray_ptr++);
            }
  printf("\n");

  for (iArray_ptr = iArray2, iIndex = 0; iIndex < SIZE; iIndex++)
            printf("iArray2[%d]= %-2d ",iIndex, *iArray_ptr++);
  printf("\n");

  for (iIndex = 0; iIndex < SIZE +1 ; iIndex++)
            printf ("iArray2[%d]= %-2d ", iIndex, *(iArray2 + iIndex));
      /* there are 4 numbers in the array, but 5 numbers are printed */
      /* this is an example of what not to do, don't do it */

  printf("\n\nThe size of iArray1 is %d bytes.", sizeof (iArray1));

  return 0;
}
/*
iArray1[0] = 10   iArray1[1] = 11   iArray1[2] = 12   iArray1[3] = 13

Enter an integer  1
Enter an integer  4
Enter an integer  2
Enter an integer  3

iArray2[0]= 1  iArray2[1]= 4  iArray2[2]= 2  iArray2[3]= 3
iArray2[0]= 1  iArray2[1]= 4  iArray2[2]= 2  iArray2[3]= 3  iArray2[4]= -858993460

The size of iArray1 is 16 bytes.

Press any key to continue . . .
*/
```

Commented [KW13]: Initialize the pointer to &iArray1[0]

Commented [KW14]: Put 10 into iArray1[0] using array notation.

Commented [KW15]: Put 11 into iArray1[1] using array notation with pointer syntax.

Commented [KW16]: Put 12 into iArray1[2] using pointer notation.

Commented [KW17]: Put 13 into iArray1[3] using array notation with a pointer.

Commented [KW18]: Notice the scanf() syntax when using pointer notation.

Commented [KW19]: Array was indexed outside of its boundaries which allowed, but NOT recommended.

```
1  /* input an array and determine the mean.
2  */
3  #define MAX  10
4  #include<stdio.h>
5  #include<conio.h>
6
7  int   read_array(int iAr[]);
8  void  show_array(int iAr[], int iN);
9  float mean(int iAr[], int iN);
10 /*-------------------------------------------------------------------------*/
11 int main(void)
12 {
13   int data[MAX], size;
14   float average;
15
16   size = read_array(data);
17   if (size == 0)
18     printf("Array size is zero.\n");
19   else
20     {
21     show_array(data, size);
22     average = mean(data,size);
23     printf("\n\nThe mean is %0.3f.\n",average);
24     }
25
26   printf("\nPush a key to return to the editor.");
27   getch();
28   return 0;
29 } /*-------------------------------------------------------------------------*/
30 int   read_array(int iAr[])
31 {
32   int iElement = 0;
33   float fTemp;
34
35   printf("Enter up to %d elements.\n"
36               "To terminate entry, enter a letter.\n",MAX);
37   while ((iElement < MAX) && (scanf("%f",&fTemp) == 1))
38     iAr[iElement++] = (int)fTemp;
39   return (iElement);
40 } /*-------------------------------------------------------------------------*/
41 void  show_array(int iAr[], int iN)
42 {
43   int iElement;
44
45   puts ("The array elements are");
46   for (iElement = 0; iElement < iN; iElement++)
47     printf("%-6d", iAr[iElement]);
48 } /*-------------------------------------------------------------------------*/
49 float mean(int iAr[], int iN)
50 {
51   int iElement;
52   float fTotal = 0.0;
53
54   for (iElement = 0; iElement < iN; iElement++)
55     fTotal += (float)iAr[iElement];
56   return ( fTotal / iN);
57 }
58
```

Commented [KW20]: The value returned is the actual number of elements entered into the array, not the size of the array.

Commented [KW21]: The while() loop ensures elements less than MAX are stored as well as the number of elements stored. The scanf() returns a 1 when a float is entered and a zero when a letter is entered since its an integer value. The (int)fTemp recasts the float data into integer data,

Commented [KW22]: Initialize the fTotal variable to 0 so the sum of the floats will be correct.

Commented [KW23]: Sum all of the array element values, divide by the number of elements, and return the average.

```
/*
Enter up to 10 elements.
To terminate entry, enter a letter.
12
23
34
56
78
90
a
The array elements are
12    23    34    56    78    90

The mean is 48.833.

Push a key to return to the editor.
*/
```