An array is a list of data objects, all of the same data type. These are stored sequentially in memory. They all have the same variable name, and are referred to individually by means of subscripts in brackets, e.g.:
X[3] = 24.6;
X[2] = 4 * A + 13;

In this case, X is a one-dimensional array.

Example:

4	-3	5	6
Y[0]	Y[1]	Y[2]	Y[3]

4.7	9.2
Z[0]	Z[1]

2	3
X[0]	X[1]

A [1] B [2] C [3] D [4]

What is the value of each of the following?

Y[0]	_____	Y[1]	_____	X[0]	_____
A	_____	Y[B]	_____	Z[A]	_____
B	_____	Y[C]	_____	Z[B]	_____
Y[A]	_____	X[A]	_____	Y[A+2]	_____

The subscript of a particular element refers to the position of that element in the array. It is a relative address. When an array is stored in memory, it is stored much this way, with the elements in contiguous storage locations.

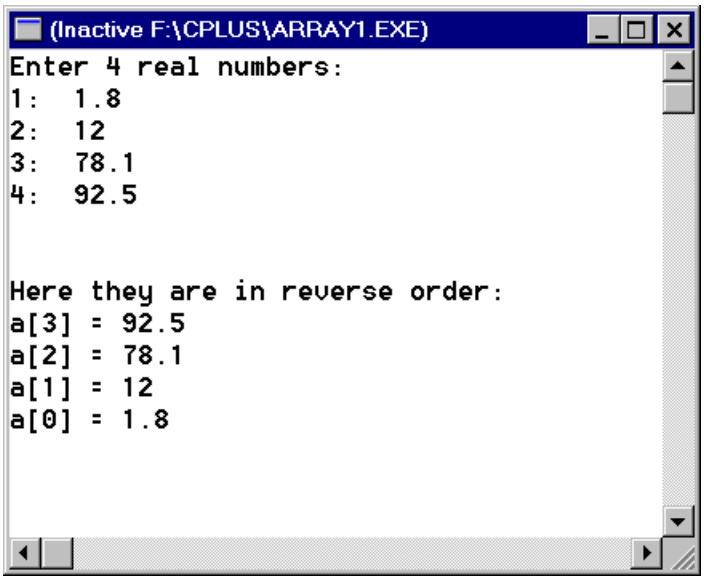
Starting the subscripts from 0 may not be totally natural to us, but it has the advantage that the subscript of a particular array element is always the same as the number of "steps" needed to jump to that element (from the beginning of the array), i.e., a[2] is 2 step away from a[0]; a[5] is 5 steps away from a[0]; etc.

Array elements may be declared on any data type, simple or structured, as long as they are all the same type.

To declare an array:
```
int X [100];
```

To access array elements we use the for loop.

```
//array1.cpp
//modified from Schaum's ex. 5.1, p. 128
#include <iostream>
main(){
   double a[4];
   cout << "Enter 4 real numbers:  \n";
   for (int i =1; i<=4; i++){
      cout << i << ":  ";
      cin >> a [i-1];
   }//end for
   cout <<"\n\nHere they are in reverse order:  \n";
   for (int i=3; i>=0; i--)
      cout << "a[" << i << "] = " << a[i] << endl;
}//end main
```



You can do it with this program too:

```
//array2.cpp
//modified from Schaum's ex. 5.1, p. 128
#include <iostream>
main(){
   const int SIZE = 4;
   double a[SIZE];
   cout << "Enter " << SIZE << " real numbers:  \n";
   for (int i=1; i<=SIZE; i++) {
      cout << i << ":  ";
      cin >> a[i-1];
   }//end for
   cout << "\n\nHere they are in reverse order:  \n";
   for (int i=SIZE-1; i>=0; i--)
      cout << "a[" << i << "] = " << a[i] << endl;
}//end main
```

Using Arrays in the Grading Assignment

What we (probably) have:

```
...
cin >> ID;
while (ID != “end”){
      cin >> LastName >> FirstName >> grade1 >> grade2 >> grade3 >> grade4;
      FinalGrade = weight1*grade1
            + weight2*grade2
            + weight3*grade3
            + weight4*grade4;
      cout << id << FirstName << LastName << grade1 << grade2 << grade3
            << grade4 << FinalGrade << endl;
      SumGrades += FinalGrade;
      n++;
      if (FinalGrade > MaxSoFar)
            MaxSoFar = FinalGrade;
      if (FinalGrade < MinSoFar)
            MinSoFar = FinalGrade;
      cin >> ID;
} //end while
...
```

Can we use a loop to compute FinalGrade? Sure (but –)

```
....
while (ID != “end”){
      cin >> LastName >> FirstName;
      FinalGrade = 0;
      for (int i=1; i<=4; i++){
            cin >> weight >> grade;
            FinalGrade += weight*grade;
      } //end for
      cout ...
```

What’s the problem here? It will work but we have to re-enter the weights for the four exams each time we read a new student’s info.

Replacing weights and grades with arrays:

```
....
while (ID != “end”){
      cin >> LastName >> FirstName;
      FinalGrade = 0;
      for (int i=1; i<=4; i++){
            cin >> grade[i];
            FinalGrade += weight[i] * grade[i];
      } //end for
      cout ...
```

To initialize an array:

```
float a[4] = {22.2, 44.4, 66.6, 88.8};    or
float a[ ] = {22.2, 44.4, 66.6, 88.8};
```

22.2	44.4	66.6	88.8
a[0]	a[1]	a[2]	a[3]

or

```
float a[4] = {22.2, 44.4};
```

22.2	44.4	0.0	0.0
a[0]	a[1]	a[2]	a[3]

or

```
float a[4] = {0.0};
```

0.0	0.0	0.0	0.0
a[0]	a[1]	a[2]	a[3]

If you omit the initialization, you may find errors in your program due to "garbage" in the array.

To access array values, we usually use the for loop, e.g.:

```
for (j=1; j<=100; j++) cin >> X[j];
for (j=1; j<=50; j++) Z[j] = X[j] + Y[j];
```

Arrays may be passed to functions as arguments.

Problem: Write an algorithm to find the minimum value in an array:

Solution:

```
min = 1;
for (j=0; j<=SIZE-1; j++)
   if (x[j] < x[min]) min = j;
```

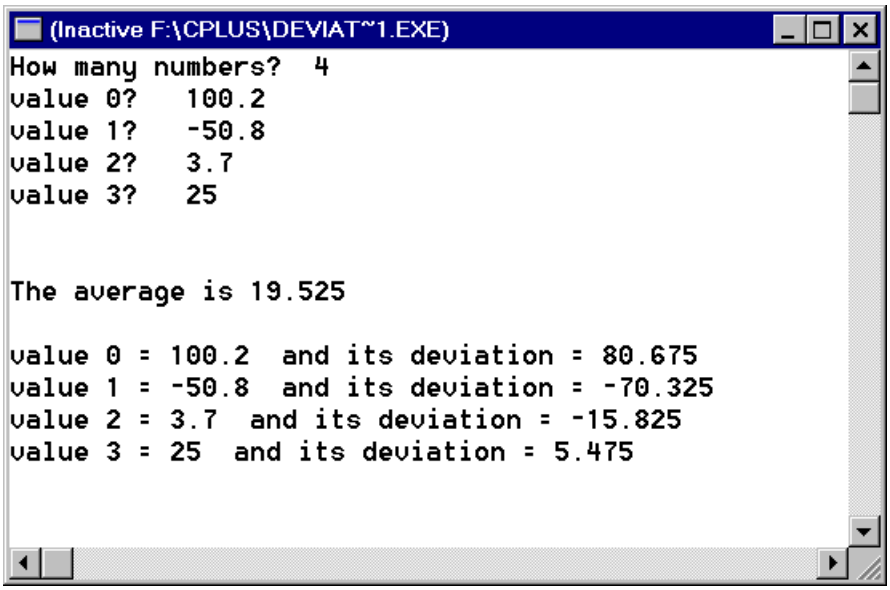
Problem: Write a program to calculate and print the average of a set of numbers, and also the deviations of each number from the average.

1st try:

```
for (int i=o; i<n; i++)
    cin >> x[i];
for (int i=o; i<n; i++)
    sum += x[i];
avg = sum / n;
```

Solution:

```
//deviations.cpp
//
#include <iostream>
main(){
   int n, count;
   double sum=0, average, deviation;
   double x[100];
   cout << "How many numbers?  ";
   cin >> n;
   for (count=0; count<=n-1; count++){
      cout << "value " << count << "?   ";
      cin >> x[count];
      sum += x[count];
   }//end for
   average = sum / n;
   cout << "\n\nThe average is " << average << endl << endl;
   for (count=0; count<=n-1; count++){
      deviation = x[count] - average;
      cout << "value " << count << " = " << x[count]
                       << "  and its deviation = " << deviation << endl;
   }//end for
}//end main
```


(Inactive F:\CPLUS\DEVIAT~1.EXE)
How many numbers? 4
value 0?   100.2
value 1?   -50.8
value 2?   3.7
value 3?   25
The average is 19.525
value 0 = 100.2 and its deviation = 80.675
value 1 = -50.8 and its deviation = -70.325
value 2 = 3.7 and its deviation = -15.825
value 3 = 25 and its deviation = 5.475

Find the error:

```
#include <iostream>
using namespace std;
const int MAX=5;
int main(){
   double x[MAX];
   x[0]=1;
   for (int i=0; i<MAX; i++)
      x[i] = i * x[i-1];
   for (int i=0; i<MAX; i++)
      cout << “X[“ << i << “]= ”<< x[i] << endl;
   return 0;
}
```

What is output?

```
include <iostream>
using namespace std;
const int MAX=5;
int main(){
   double x[MAX];
   for (int i=0; i<MAX; i++)
      x[i] = i * i;
   for (int i=0; i<MAX; i++)
      cout << “X[“ << i << “]”= << x[i] << endl;
   return 0;
}
```

Problem: Write an algorithm to find the minimum value in an array:

Solution #1:

```
min = 999;              // or some other very large value
for (int j=0; j<SIZE; j++)
   if (x[j] < min) min = x[j];
```

Solution #2:

```
min = x[0];             // min initialized to the first array element
for (int j=0; j<SIZE; j++)
   if (x[j] < min) min = x[j];
```

Solution #3:

```
min = x[0];             // min initialized to the first array element
for (int j=1; j<SIZE; j++)       //start comparing at x[1]
   if (x[j] < min) min = x[j];
```

Solution #4:

```
min = 0;                //min tells WHERE, not WHAT the minimum is
for (int j=1; j< SIZE; j++)
   if (x[j] < x[min]) min = j;
```

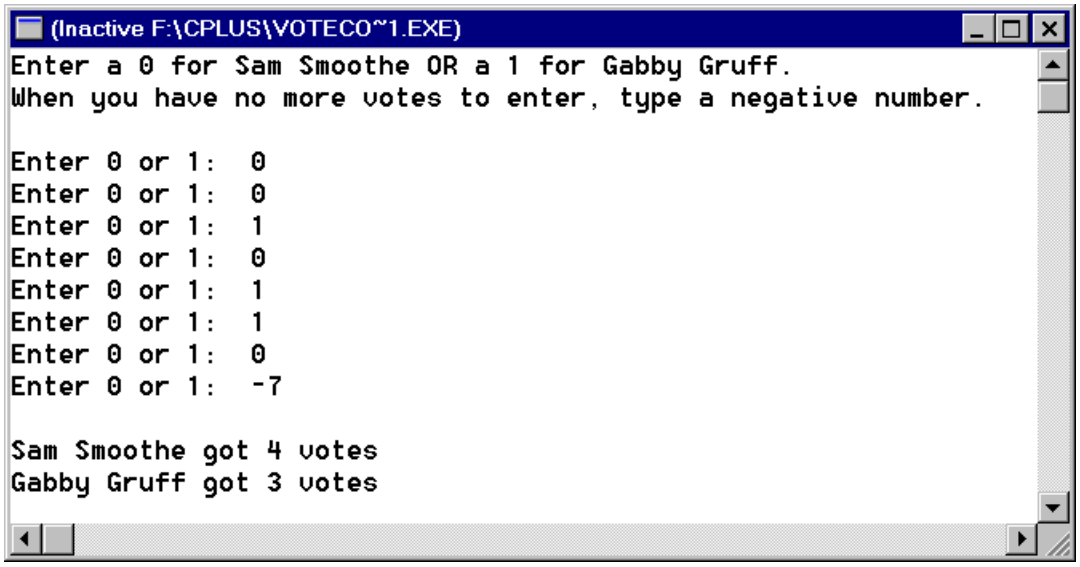
Problem: Write a program to count the votes in a local community election.

1st try:

```
while (v>=0)
   if (v==1) votes[1]++
   else if (v==2) … … …
```

Solution:

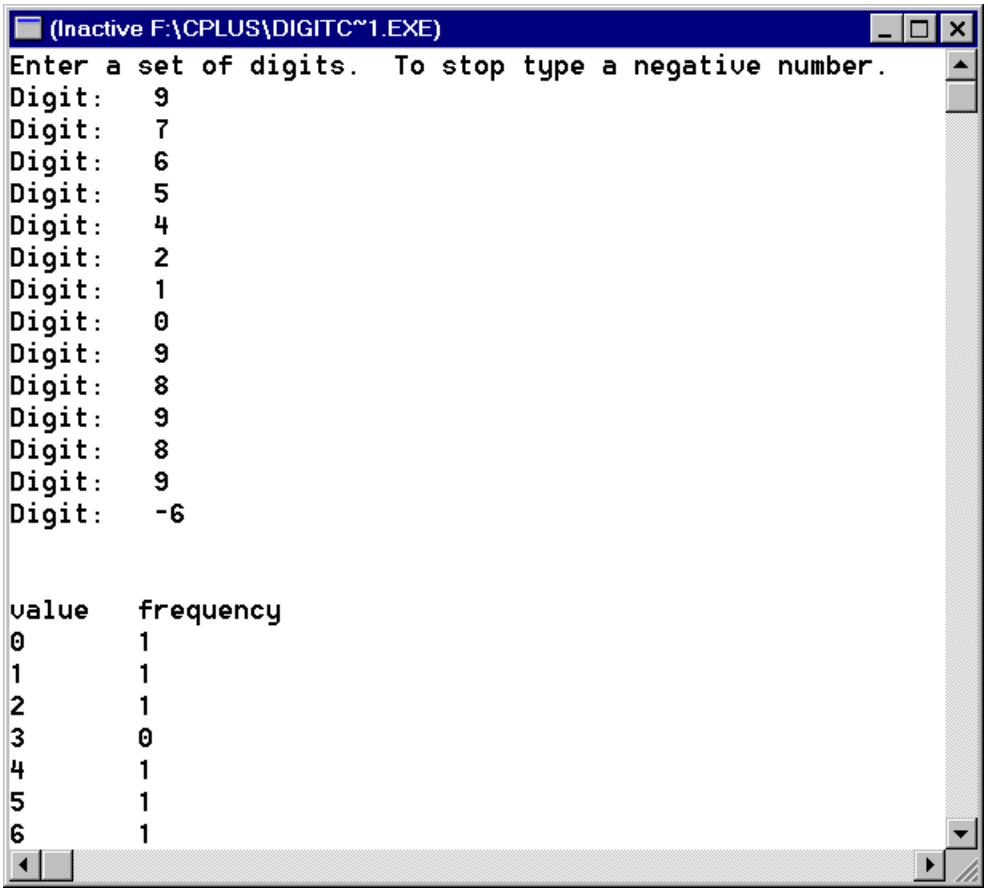
```
//votecounter.cpp
//
#include <iostream>
int main(){
   short v;
   int votes[2] = {0};
   cout << "Enter a 0 for Sam Smoothe OR a 1 for Gabby Gruff.\n"
      << "When you have no more votes to enter, "
      << "type a negative number.\n\n"
      << "Enter 0 or 1:  ";
   cin >> v;
   while (v >= 0) {
      votes[v]++;
      cout << "Enter 0 or 1:  ";
      cin >> v;
   }//end while
   cout << "\nSam Smoothe got " << votes[0] << " votes\n";
   cout << "Gabby Gruff got " << votes[1] << " votes\n";
   return 0;
}//end main
```



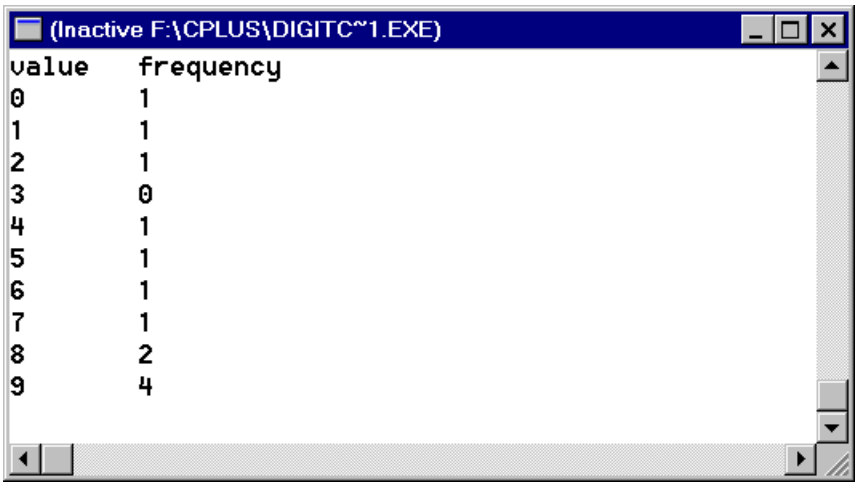
Problem: Write a program to count and print the frequencies of the integers 0 - 9 entered as data.

Solution:

```
//digitcount.cpp
//
#include <iostream>
int main(){
   short i;
   int freq[10] = {0};
   cout << "Enter a set of digits.  To stop type a negative number."
      << "\nDigit:   ";
   cin >> i;
   while (i >= 0){
      freq[i] ++;
      cout << "Digit:   "; cin >> i;
   }
   cout << "\n\nvalue \tfrequency \n";
   for (i=0; i<=9; i++)
      cout << i << '\t' << freq[i] << endl;
}//end main
```



output continued on next page…



Arrays may be passed to functions as arguments….

Arrays and Functions

When an array is passed to a function, only the name and type of the array is passed. The name of a variable is equivalent to a storage location, so the effect is that the array is passed by reference even if it looks like it's being passed by value (no '&').

Example: Using functions to process an array of data.

```
//sum.cpp
// Reads an array of data values, prints it, and sums the values.

#include <iostream>
const int SIZE = 20;
void readarray (float[], int&);
void printarray (const float[], int);
float sumarray (const float[], int);
//*********************************************************************
int main(){
   int n;
   float data [SIZE] = {0.0};
   readarray(data, n);
   printarray(data, n);
   cout << "The sum is  " << sumarray(data, n);
   cout << "\n\nPress any key to close window."; char c; cin >> c;
   return 0;
}
//*********************************************************************
void readarray(float x[], int &n){
   n = 0;   // n is the "number of numbers" in the array
   char more='y';
   while (more != 'n' && more != 'N'){
      cout << "\nEnter a number:  "; cin >> x[n];
      n++;
      cout << "Another number? (y or n)  ";
      cin >> more;
      }
}
//*********************************************************************
float sumarray (const float x[], int n){
   float sum = 0;
   for (int i=0; i<n; i++) sum += x[i];
   return sum;
}
//*********************************************************************
void printarray (const float x[], int n){
   cout << "The data values are:\n";
   for (int i=0; i<n; i++) cout << '\t' << x[i];
   cout << endl;
}
```

When the function does not have to change the values in the array, the keyword **const** ensures that this will not happen inadvertently.

Since only the name of the array is passed, not the size (or even all the values stored in the array), we also need to specify an additional parameter - the number of array elements to be processed. We can avoid this extra parameter by specifying the size of the array, e.g.,

```
int sumarray (int x[100]);
```

but that makes the function much less general.