Arrays

If *T* is any C++ type except **void** or "function returning …," the type "array of *T*" may be declared. Values of this type are sequences of elements of type *T*. All arrays are 0-origin.

Example

int *F***[3];** //*F* is an array of type **int**. Array *F* has elements *F***[0]**, *F***[1]**, and *F***[2]**.

```
#include <iostream>
using namespace std;
int main()
{    int F[10];
     F[0]=0;
     F[1]=1;
     for (int i=2;i<10;i++) F[i]=F[i-1]+F[i-2];
     for (int i=0;i<10;i++) cout << F[i] << " ";
     cout << endl;
     return 0;
}
```

Figure 1. Program **p01**, an array of ten integers assigned values from a fibonacci sequence

Program **p01** prints.

```
0 1 1 2 3 5 8 13 21 34
```

Program **p01** notes.

1. Variable *F* is an array having ten (10) integer elements.
2. The first two elements, elements 0 and 1, *F***[0]** and *F***[1]** are initialized to the first two values in a fibonacci sequence.
3. The first *for-statement* assigns fibonacci values to remaining elements of array *F*.
4. The second *for-statement* prints the fibonacci sequence.

Multiple dimensions. Arrays can have multiple dimensions. For example.

double *A*[3][5]; //*A* is an array having three rows and five columns.

```
#include <iostream>
#include <ctime>
#include <cmath>
#include <iomanip>
using namespace std;
class Uniform {
public:
    Uniform()
    {   time_t t;
        srand((unsigned)time(&t));
    }
    double Sample(void) {return (double)rand()/RAND_MAX;}
};

int main()
{   double A[3][5];
    Uniform U;
    for (int r=0;r<3;r++) {
        for (int c=0;c<5;c++) {
            A[r][c]=U.Sample();
        }
    }
    for (int r=0;r<3;r++) {
        cout << endl;
        for (int c=0;c<5;c++) {
            cout << setw(10) << fixed << setprecision(4) << A[r][c];
        }
    }
    cout << endl;
    return 0;
}
```

Figure 2. Program **p02,** a two-dimensional array having values taken from the uniform distribution.

Program **p02** prints.

```
    0.3790    0.4577    0.3032    0.7310    0.4557
    0.5337    0.7066    0.1650    0.3751    0.2901
    0.6194    0.4795    0.5347    0.3802    0.8400
```

Array initialization.

An array can be initialized by a list of values. For example:

```
int A[]={1, 2, 3, 4};          //Array A has four elements, A[0]==1,A[1]==2,A[2]==3, A[3]==4
char T[]={'t','e','x','t'};  //Array T has four elements, T[0]=='t',T[1]=='e',T[2]=='x',T[3]=='t'
```

When an array is declared without a specific size, but with an initializer list, the size is calculated by counting the elements of the initializer list. Consequently *A* and *T* have an implied declaration of **int** *A***[4];** and **char** *T***[4];**

If a size is explicitly specified, it is an error to give surplus elements in an initializer.

```
double f[2]={1.602e-19,9.1095e-31,6.022e23};          //error; too many initializers
```

If the initializer has too few elements to completely fill the array, remaining elements are assigned a value of zero. For example:

```
int v[8]={1, 2, 3, 4 };
```

is equivalent to

```
int v[8]={1, 2, 3, 4, 0, 0, 0, 0 };
```

Pointers are arrays.

The name of the array without a subscript is a pointer to the first element of the array. For example:

```
int A[]={5,2,1}; cout << *A;          //5, the value of A[0] is printed
```

The address of an element of an array is computed by adding the subscript expression to the base of the array, which is the array name. For example:

```
cout << A[1];
```

is equivalent to

```
cout << *(A+1)
```

Elements of an array can be referenced by separately declared pointers. For example:

```
int A[]={5,2,1,0,3};          //A is an array having five integer elements
int* p=A;                     //Integer pointer p points to A[0];
cout << *++p;                 //2, the value of A[1] is printed.
p=p+2;                        //p now points to A[3].
cout << *p;                   //0, the value of A[3] is printed
```

The foregoing example is illustrated in figure 3.

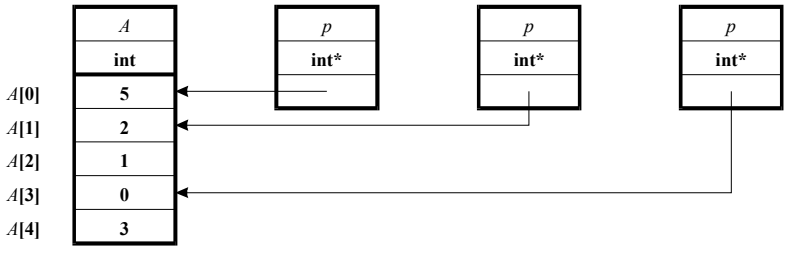

Figure 3. Array *A* and integer pointer *p*.

Arrays as parameters and arguments.

The name of the array is passed as an argument.

```
const int N=17;
int A[N];
ArrayInit(A,N);
```

The name of the array following by [] appears in the parameter declaration.

```
void ArrayInit(int A[],int N);
```

Since only the address of the first element is passed, it is a good idea to define a second parameter that bounds the array. Integer parameter *N* contains the number of elements in the array.

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;
void ArrayInit(int A[],int N)
{    srand(29);
     for (int i=0;i<N;i++) A[i]=rand();
}
void ArrayPrint(int A[],int N)
{    for (int i=0;i<N;i++) {
          if (i%3==0) cout << endl;
          cout << setw(15) << A[i];
     }
     cout << endl;
}
int main()
{    const int N=17;
     int A[N];
     ArrayInit(A,N);
     ArrayPrint(A,N);
     return 0;
}
```

Figure 4. Program **p04**, array parameters and arguments**.**

Program **p04** prints.

```
     1881172855       991966287     1624772004
      564079445     1216093759      815928867
      955647895     1140828572     1784489259
     1988587541     1600840080      347140191
      220585252     2076303548      466324046
     1690747329      484526685
```