Arrays

Random access lists of elements

Pallab Dasgupta
Professor,
Dept. of Computer Sc & Engg

Basics of Arrays

Array

Many applications require multiple data items that have common characteristics.

- **In mathematics, we often express such groups of data items in indexed form:**
  - $x_1, x_2, x_3, \ldots, x_n$

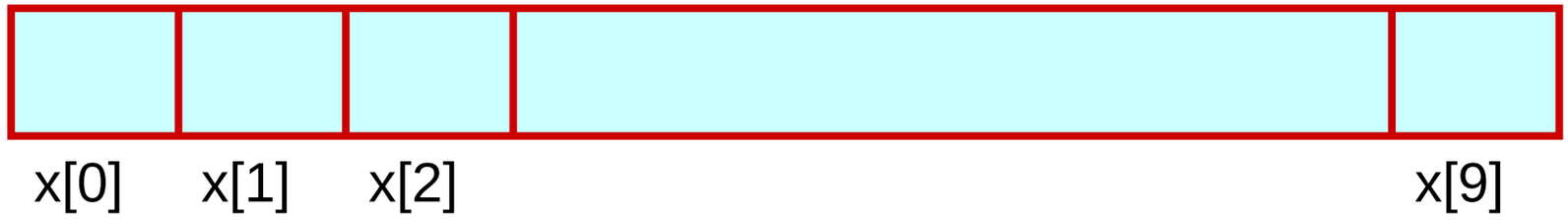
Array is a data structure which can represent a collection of data items which have the same data type (float / int / char)

Using Arrays

All the data items constituting the group share the same name.

```
int  x[10];
```

Individual elements are accessed by specifying the index.



X is a 10-element one dimensional array

Declaring Arrays

Like variables, arrays must be declared before they are used.

General syntax:

```
type   array-name [size];
```

- **type** specifies the data type of the array elements (int, float, char, etc.)
- **size** is an integer constant representing the number of elements that can be stored in the array.

```
int   marks[5];
```

- **marks** is an array containing a maximum of 5 integers.

Examples:

```
int  x[10];
char  line[80];
float  points[150];
char  name[35];
```

If we are not sure of the exact size of the array, we can define an array of a large size.

```
int   marks[50];
```

though in a particular run we may only be using, say, 10 elements.

How is an array stored in memory?

Starting from a given memory location, the successive array elements are allocated space in consecutive memory locations.

- **x: starting address of the array in memory**
- **k: number of bytes allocated per array element**

a[i] is allocated memory location at address x + i*k

Accessing Array Elements

A particular element of the array can be accessed by specifying two things:

- Name of the array.
- Index (relative position) of the element in the array.

In C, the index of an array starts from zero.

Example:

- An array is defined as `int x[10];`
- The first element of the array x can be accessed as x[0], fourth element as x[3], tenth element as x[9], etc.

Contd.

The array index must evaluate to an integer between 0 and *n* – 1 where *n* is the number of elements in the array.

```
a[ x+2 ] =  25;
b[ 3*x – y ]  =  a[ 10 – x ] + 5;
```

A Warning

In C, while accessing array elements, array bounds are not checked.

Example:

```
int   marks[5];
:
:
marks[8] = 75;
```

- **This may cause a *segmentation fault* at runtime.**
  - ***Not always !!***
  - **It may also result in unpredictable program results**

Initialization of Arrays

General form:

```
type   array_name[size]  =  { list of values };
```

Examples:

```
int  marks[5]  = { 72, 83, 65, 80, 76 };
char  name[4] = { 'A', 'm', 'i', 't' };
```

Some special cases:

- If the number of values in the list is less than the number of elements, the remaining elements are automatically set to zero.

```
float  total[5] =  { 24.2,  –12.5, 35.1}
```

causes the following assignments: total[0] = 24.2 total[1] = –12.5 total[2] = 35.1 total[3] = 0 total[4]= 0

Contd.

- The size may be omitted. In such cases the compiler automatically allocates enough space for all initialized elements.

```
int   flag[ ] = {1, 1, 1, 0};
char  name[ ] = {‘A’, ‘m’, ‘i’, ‘t’};
```

Character Arrays and Strings

char C[8] = { 'a', 'b', 'h', 'i', 'j', 'i', 't', '\0' };

The last (7th) location receives the null character '\0'.

Null-terminated character arrays are also called strings.

Strings can be initialized in an alternative way.

The last declaration is equivalent to: char C[8] = "abhijit";

The trailing null character is missing here. C automatically puts it at the end.

Note also that for individual characters, C uses single quotes, whereas for strings, it uses double quotes.

Examples

Finding the minimum of a set of 10 numbers

```
#include  <stdio.h>
main()
{
        int  a[10], i, min;

        for  (i=0; i<10; i++) scanf ("%d", &a[i]);

        min = a[0];
        for  (i=1; i<10; i++)
        {
                if  (a[i] < min) min = a[i];
        }
        printf ("\n Minimum is %d", min);
}
```

Slightly modified version

```
#define SIZE 100
#include  <stdio.h>
main()
{
        int  a[SIZE], i, min, N;

        scanf("%d", &N);
        for  (i=0; i < N; i++) scanf ("%d", &a[i]);

        min = a[0];
        for  (i=1; i < N; i++)
        {
                if  (a[i] < min) min = a[i];
        }
        printf ("\n Minimum is %d", min);
}
```

Computing GPA

Grade points received in the different subjects

Credits of the different subjects

```
#define SUBJECTS 6
#include <stdio.h>

main( )
{
   int grade_pt[SUBJECTS], credit[SUBJECTS], k, gp_sum=0, cred_sum=0, GPA;

   for ( k=0; k < SUBJECTS; k++ ) scanf ("%d %d", &grade_pt[k], &cred[k]);

   for ( k=0; k < SUBJECTS; k++ )
   {
         gp_sum += grade_pt[ k ] * credit[ k ];
         cred_sum += credit[ k ];
   }
   GPA = gp_sum / cred_sum;
   printf ("\n Grade point average: is %d", GPA);
}
```

Things you cannot do

You cannot do the following:

- **Use = to assign one array a to another array b**

  ```
  a = b;
  ```

- **Use == to directly compare array variables**

  ```
  if  (a = = b)  ……..
  ```

- **Directly scanf or printf arrays**

  ```
  printf (“……”, a);
  ```

How to copy the elements of one array to another?

By copying individual elements:

```
for  ( j=0 ; j<25 ; j++ )  a[ j ] = b[ j ];
```

How to read the elements of an array?

By reading them one element at a time

```
for  ( j=0 ; j<25 ; j++ ) scanf ( “%f”, &a[ j ] );
```

The ampersand (&) is necessary.

The elements can be entered all in one line or in different lines.

How to print the elements of an array?

By printing them one element at a time.

```
for  ( j=0 ; j<25 ; j++ ) printf ( "\n %f", a[ j ] );
```

- The elements are printed one per line.

```
for  ( j=0 ; j<25 ; j++ ) printf ( " %f", a[ j ] );
```

- The elements are printed all in one line (starting with a new line).

Arrays and Functions

Passing Arrays to Function

Array element can be passed to functions as ordinary arguments.

```
isFactor ( x[ k ], x[ 0 ] )

sin ( x[ 5 ] )
```

Passing an entire array to a function

An array name can be used as an argument to a function.

- **Permits the entire array to be passed to the function.**
- **The way it is passed differs from that for ordinary variables.**

Rules:

- **The array name must appear by itself as argument, without brackets or subscripts.**
- **The corresponding formal argument is written in the same manner.**
    - **Declared by writing the array name with a pair of empty brackets.**

Array as input parameter

```
#define ASIZE 5
float average ( int a[ ] )      {
        int k, total=0;
        for ( k=0; k<ASIZE;  k++ )
                total = total + a[ k ];
        return ( (float) total  / (float) ASIZE );
}

main ( )  {
    int x[ASIZE]  = {10, 20, 30, 40, 50}; float xavg;
        xavg = average (x) ;
}
```

Modified – variable number of elements in the array

```
#define ASIZE 100
float average ( int a[ ], int N )  {
        int k, total=0;
        for ( k=0; k<N;  k++ )
                total = total + a[ k ];
        return ( (float) total  / (float) ASIZE );
}

main ( )  {
    int x[ASIZE]  = {10, 20, 30, 40, 50};
    int Z = 5; float xavg;
        xavg = average (x, Z) ;
}
```

Note that the size of the array is not passed as a parameter.

But we need to pass N, the actual number of data in the array.

Arrays can also be used as Output Parameters

```
void VectorSum (int a[ ], int b[ ], int vsum[ ], int length)  {
        int i;
        for (i=0; i<length; i=i+1) vsum[ i ] = a[ i ] + b[ i ] ;
}
void PrintVector (int a[ ], int length)  {
        int i;
        for (i=0; i<length; i++) printf ("%d ", a[i]);
}
int main (void)    {
        int x[3] = {1,2,3},  y[3] = {4,5,6},  z[3];
        VectorSum (x, y, z, 3);
        PrintVector (z, 3) ;
}
```

The Actual Mechanism

When an array is passed to a function, the values of the array elements are not passed to the function.

- The array name is interpreted as the address of the first array element.
- The formal argument therefore becomes a pointer to the first array element.
- When an array element is accessed inside the function, the address is calculated using the formula stated before.
- Changes made inside the function are thus also reflected in the calling program.

Passing arrays as parameters

Passing parameters in this way is called

call-by-reference.

Normally parameters are passed in C using

call-by-value.

Basically:

- If a function changes the values of array elements, then these changes will be made to the original array that is passed to the function.
- This does not apply when an individual element is passed on as argument.

Examples

A function for reversing an array

Reversing A = { 1, 2, 0, 5, 3 } results is A = { 3, 5, 0, 2, 1 }

```
void reverse( int x[ ], int n )
{           int k, temp;
            for ( k=0;  k < n;  k++ ) {
                        temp = x[ k ] ;
                        x[ k ] = x[ n – k – 1 ] ;
                        x[ n – k – 1 ] = temp;
            }
}
```

What's wrong in this code?

Bubble Sort

Sorting A = { 1, 2, 0, 5, 3 } results is A = { 0, 1, 2, 3, 5 }

```
void bubblesort( int x[ ], int n )
{           int  j, k, temp;

            for ( j=0; j < n – 1; j++ )
                        for ( k=0;  k < n – j – 1;  k++ ) {

                                    if ( x[ k ] > x[ k + 1] ) {
                                                temp = x[ k ] ;  x[ k ] = x[ n – k – 1 ] ; x[ n – k – 1 ] = temp;
                                    }
                        }
            }
}
```

Finding the largest contiguous sequence of equal numbers

The largest sequence of equal numbers in A = { 1, 1, 2, 2, 0, 0, 0, 1, 1, 5, 3 } is 0, 0, 0 (A[4] – A[6])

```
k = 0;   maxbegin = 0;    maxcount = 1;
while ( k < N )
{
        ssbegin = k;  count = 1;
        while ( x[ k ] == x[ k+1 ] ) {
                k++; count++;
                if ( k == N – 1 ) break;
        }
        if ( count > maxcount ) { maxbegin = ssbegin; maxcount = count; }
        k++;
}
printf( “Sequence starting from x[ %d ] = “, maxbegin );
for ( k=0; k < maxcount; k++ ) printf(“%d, “, x[k] ); printf (“\n”);
```