Arrays

CS10001: Programming & Data Structures

Pabitra Mitra

Dept. of Computer Sc. & Engg.,
Indian Institute of Technology Kharagpur

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)**

Example: Finding Minima of Numbers

3 numbers

```
if   ((a <= b) && (a <= c))
   min = a;
else
   if   (b <= c)
      min = b;
   else
      min = c;
```

4 numbers

```
if   ((a <= b) && (a <= c) && (a <= d))
   min = a;
else
   if   ((b <= c) && (b <= d))
      min = b;
   else
      if  (c <= d)
         min = c;
      else
         min = d;
```
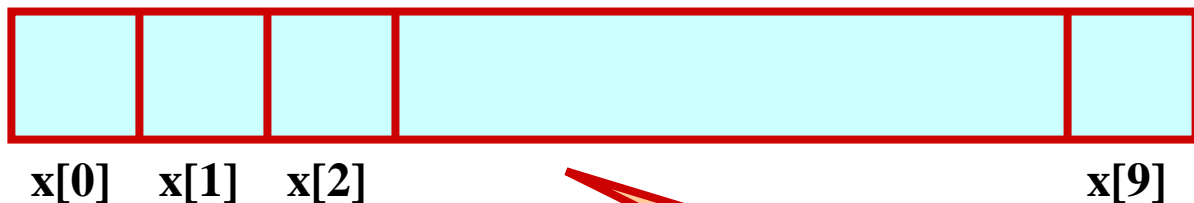
The Problem

- **Suppose we have 10 numbers to handle.**
- **Or 20.**
- **Or 100.**
- **Where do we store the numbers ? Use 100 variables ??**
- **How to tackle this problem?**

- **Solution:**
  - **Use arrays.**

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[0]   x[1]   x[2]   x[9]

X is a 10-element one dimensional array

Declaring Arrays

- **Like variables, the arrays that are used in a program must be declared before they are used.**

- **General syntax:**

type   array-name [size];

  - **type specifies the type of element that will be contained in the array (int, float, char, etc.)**
  - **size is an integer constant which indicates the maximum number of elements that can be stored inside 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.

7

How an array is stored in memory?

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

Array a

- **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.**

9

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;
  ```

  - **The above assignment would not necessarily cause an error.**
  - **Rather, it may 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};

        ➔ **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' };

- C[0] gets the value 'a', C[1] the value 'b', and so on. 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.

Example 1: Find the minimum of a set of 10 numbers

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

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

   min = 99999;
   for  (i=0; i<10; i++)
   {
      if  (a[i] < min)
         min = a[i];
   }
   printf ("\n Minimum is %d", min);
}
```

Alternate Version 1

Change only one line to change the problem size

```c
#include  <stdio.h>
#define   size   10

main()
{
    int  a[size], i, min;

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

    min = 99999;
    for  (i=0; i<size; i++)
    {
        if  (a[i] < min)
            min = a[i];
    }
    printf ("\n Minimum is %d", min);
}
```

Alternate Version 2

Define an array of large size and use only the required number of elements

```c
#include  <stdio.h>

main()
{
   int  a[100], i, min, n;

   scanf ("%d", &n);  /* Number of elements */
   for  (i=0; i<n; i++)
      scanf ("%d", &a[i]);

   min = 99999;
   for  (i=0; i<n; i++)
   {
      if  (a[i] < min)
         min = a[i];
   }
   printf ("\n Minimum is %d", min);
}
```

Example 2: Computing gpa

Handling two arrays at the same time

```c
#include  <stdio.h>
#define  nsub  6

main()
{
    int  grade_pt[nsub], cred[nsub], i,
        gp_sum=0, cred_sum=0, gpa;

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

    for  (i=0; i<nsub; i++)
    {
        gp_sum += grade_pt[i] * cred[i];
        cred_sum += cred[i];
    }
    gpa = gp_sum / cred_sum;
    printf ("\n Grade point average:  is %d", gpa);
}
```

Things you cannot do

- **You cannot**
  - **use = to assign one array variable to another**

    a = b;   /* a and b are arrays */

  - **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.**

    printf ("\n");

    for (j=0; j<25; j++)

        printf (" %f", a[j]);

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

Two Dimensional Arrays

- **We have seen that an array variable can store a list of values.**

- **Many applications require us to store a table of values.**

	Subject 1	Subject 2	Subject 3	Subject 4	Subject 5
Student 1	75	82	90	65	76
Student 2	68	75	80	70	72
Student 3	88	74	85	76	80
Student 4	50	65	68	40	70

Contd.

- **The table contains a total of 20 values, five in each line.**
  - The table can be regarded as a **matrix** consisting of **four rows** and **five columns**.
- **C allows us to define such tables of items by using two-dimensional arrays.**

Declaring 2-D Arrays

- **General form:**

  type   array_name [row_size][column_size];

- **Examples:**

  int marks[4][5];

  float sales[12][25];

  double matrix[100][100];

Accessing Elements of a 2-D Array

- **Similar to that for 1-D array, but use two indices.**
  - **First indicates row, second indicates column.**
  - **Both the indices should be expressions which evaluate to integer values.**
- **Examples:**

  x[m][n] = 0;

  c[i][k] += a[i][j] * b[j][k];

  a = sqrt (a[j*3][k]);

How is a 2-D array is stored in memory?

- **Starting from a given memory location, the elements are stored row-wise in consecutive memory locations.**
  - x: starting address of the array in memory
  - c: number of columns
  - k: number of bytes allocated per array element
  - a[i][j] ➔ is allocated memory location at

    address $x + (i * c + j) * k$

a[0]0] a[0][1] a[0]2] a[0][3]	a[1][0] a[1][1] a[1][2] a[1][3]	a[2][0] a[2][1] a[2][2] a[2][3]
Row 0	Row 1	Row 2

How to read the elements of a 2-D array?

- **By reading them one element at a time**

```
for  (i=0; i<nrow; i++)
    for  (j=0; j<ncol; j++)
        scanf  ("%f", &a[i][j]);
```

- **The ampersand (&) is necessary.**
- **The elements can be entered all in one line or in different lines.**

How to print the elements of a 2-D array?

- **By printing them one element at a time.**

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

  – **The elements are printed one per line.**

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

  – **The elements are all printed on the same line.**

Contd.

```
for  (i=0; i<nrow; i++)
{
    printf  ("\n");
    for  (j=0; j<ncol; j++)
        printf ("%f   ", a[i][j]);
}
```

– **The elements are printed nicely in matrix form.**

Example: Matrix Addition

```c
#include  <stdio.h>

main()
{
    int  a[100][100], b[100][100],
           c[100][100], p, q, m, n;

    scanf ("%d %d", &m, &n);

    for  (p=0; p<m; p++)
        for  (q=0; q<n; q++)
            scanf ("%d", &a[p][q]);

    for  (p=0; p<m; p++)
        for  (q=0; q<n; q++)
            scanf ("%d", &b[p][q]);

    for  (p=0; p<m; p++)
        for  (q=0; q<n; q++)
            c[p]q] = a[p][q] + b[p][q];

    for  (p=0; p<m; p++)
    {
        printf ("\n");
        for  (q=0; q<n; q++)
            printf ("%f   ", a[p][q]);
    }
}
```

Some Exercise Problems to Try Out

- **Find the mean and standard deviation of a set of n numbers.**

- **A shop stores n different types of items. Given the number of items of each type sold during a given month, and the corresponding unit prices, compute the total monthly sales.**

- **Multiple two matrices of orders mxn and nxp respectively.**

Passing Arrays to Function

- **Array element can be passed to functions as ordinary arguments.**
  - **IsFactor (x[i], x[0])**
  - **sin (x[5])**

Passing 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.

Whole array as Parameters

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

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

35

Contd.

We don't need to write the array size. It works with arrays of any size.

```c
main()
{
   int  n;
   float   list[100], avg;
   :
   avg  =  average (n, list);
   :
}

float  average  (a, x)
int  a;
float  x[];
{
   :
   sum = sum + x[i];
}
```

36

Arrays as Output Parameters

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

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.**

Contd.

- **Passing parameters in this way is called call-by-reference.**
- **Normally parameters are passed in C using call-by-value.**
- **Basically what it means?**
  - **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.**

Passing 2-D Arrays

- **Similar to that for 1-D arrays.**
  - The array contents are not copied into the function.
  - Rather, the address of the first element is passed.

- **For calculating the address of an element in a 2-D array, we need:**
  - The starting address of the array in memory.
  - Number of bytes per element.
  - Number of columns in the array.

- **The above three pieces of information must be known to the function.**

Example Usage

```
#include  <stdio.h>

main()
{
   int  a[15][25],  b[15]25];
   :
   :
   add (a, b, 15, 25);
   :
}
```

```
void  add (x, y, rows, cols)
int  x[][25], y[][25];
int  rows, cols;
{
    :
}
```

We can also write

int x[15][25], y[15][25];