Arrays

CS10001: Programming & Data Structures

Sudeshna Sarkar
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?

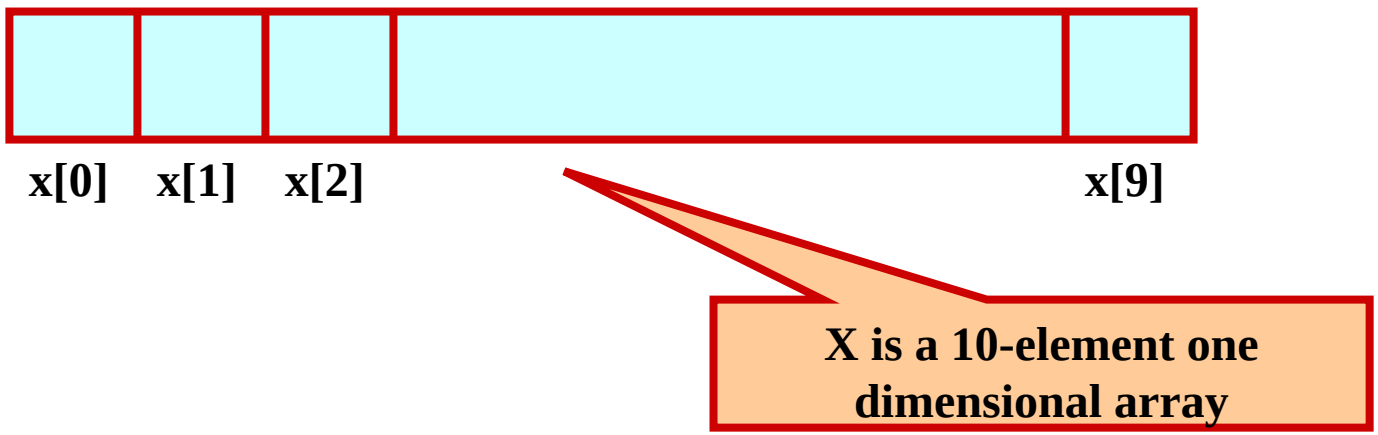
- Solution:
  - Use arrays.

Using Arrays

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

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

- Individual elements are accessed by specifying the index.



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

int marks[5];

- **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.**

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.

Index Rule

- **An array index must evaluate to an int between 0 and n-1 where n is the number of elements in the array.**
  - **marks[76]**
  - **marks[i*2+k]** **// provided i*2+k is between 0 1nd 99**

C Array bounds are not checked

```
#define S 100
marks[S] = 10;
```

```
if (0<=i && i<100)
        marks[i] = 10;
else printf ("Array index %d
        out of range", i);
```

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.

Use

- **An array element can be used wherever a simple variable of the same type can be used.**
  - **Examples :**

```
scanf (“%d”, &marks[i]);
marks[i] = (int) (sqrt(21.089));
```

Things you can and can't do

- You can not
  - use = to assign one array variable to another
  - use == to directly compare array variables
  - directly scanf or printf arrays
- But you can do these things on array elements.
- You can write functions to do them.

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

```
#include  <stdio.h>
int 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

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

Int 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

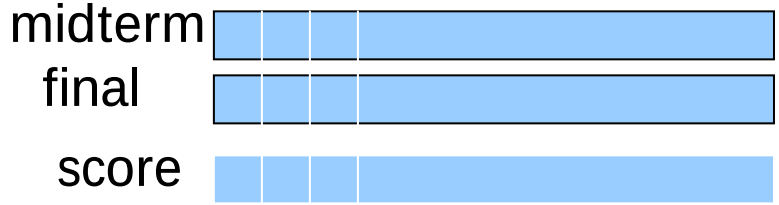
Define an array of large size and use only the required number of elements

```
#include  <stdio.h>
int 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);
}
```

- **Are Arrays necessary to solve the previous problem ?**

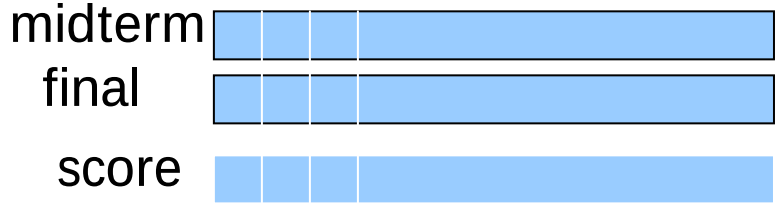
- **What about this problem :**
  - **read student marks, print all marks above average only.**

Parallel Arrays




```
/* Suppose we have input the value of NumStudents, read student i's
     grades for midterm and final, and stored them in midterm[i] and
     final[i]
   Store a weighted average in the array score */
#define MtWeight 0.3
#define FinalWeight 0.7
#define MaxStudents 100

int NumStudents;
int midterm[MaxStudents];
int final[MaxStudents];
double score[MaxStudents];
if (NumStudents < MaxStudents)
        for (i=0; i<NumStudents; i++) {
            score[i] = MtWeight* (double) midterm[i] +FinalWeight* (double) final[i];
}
```

Parallel Arrays




```
/* Suppose we have input the value of NumStudents, read student i's grades for midterm and final,
      and stored them in midterm[i] and final[i]
   Store a weighted average in the array score */
#define MtWeight 0.3
#define FinalWeight 0.7
#define MaxStudents 100

int NumStudents;
int midterm[MaxStudents];
int final[MaxStudents];
double score[MaxStudents];

if (NumStudents < MaxStudents)
        for (i=0; i<NumStudents; i++) {
            score[i] = MtWeight* (double) midterm[i] +FinalWeight*
    (double) final[i];
}
```

Computing gpa

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

int 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).**