Arrays

Indranil Sen Gupta

Dept. of Computer Science & Engg.
Indian Institute of Technology
Kharagpur

Basic Concept

- Many applications require multiple data items that have common characteristics.
  - In mathematics, we often express such groups of data items in indexed form:

    $x_1, x_2, x_3, \ldots, x_n$
- Why are arrays essential for some applications?
  - Take an example.
  - Finding the minimum of a set 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.

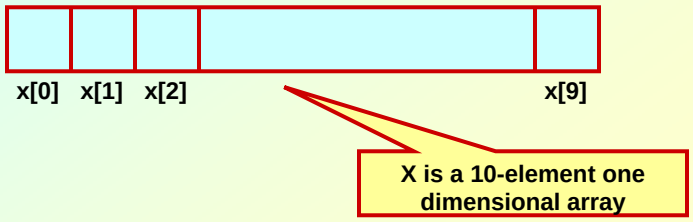
- How to tackle this problem?
- Solution:
  - Use arrays.

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.
- Example: `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];
```

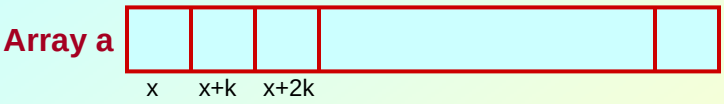
- **If we are not sure of the exact size of the array, we can define an array of a large size.**

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

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

How an array is 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

– **Element a[i] :: allocated memory location at address x + i*k**

- **First array index assumed to start at zero.**

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;
```

  - 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'};
```


Example 1: Find 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 = 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

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>

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

```
#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 GPA 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:

```
int a[25], b[25];
……
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.

```
int a[25];
……
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]);
```

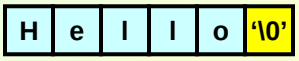
  - The elements are printed all in one line (starting with a new line).

Character String

Introduction

- A string is an array of characters.
  - Individual characters are stored in memory in ASCII code.
  - A string is represented as a sequence of characters terminated by the null ('\0') character.

"Hello" ➔



Declaring String Variables

- A string is declared like any other array:

```
char string-name [size];
```

  - size determines the number of characters in string_name.
- When a character string is assigned to a character array, it automatically appends the null character ('\0') at the end of the string.
  - size should be equal to the number of characters in the string plus one.

Examples

```
char  name[30];
char  city[15];
char  dob[11];
```

- A string may be initialized at the time of declaration.

Equivalent

```
char   city[15] = "Calcutta";
char   city[15] = {'C', 'a', 'l', 'c', 'u',
                   't', 't', 'a'};
char   dob[] = "12-10-1975";
```


Reading Strings from the Keyboard

- Two different cases will be considered:
  - Reading words
  - Reading an entire line

Reading "words"

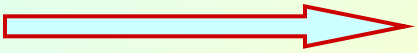
- scanf can be used with the "%s" format specification.

```
char   name[30];
 :
 :
scanf ("%s", name);
```

  - The ampersand (&) is not required before the variable name with "%s".
  - The problem here is that the string is taken to be upto the first white space (blank, tab, carriage return, etc.)
    - If we type "Rupak Biswas"
    - name will be assigned the string "Rupak"

Reading a "line of text"

- In many applications, we need to read in an entire line of text (including blank spaces).
- We can use the getchar() function for the purpose.



```
char   line[81], ch;
int  c=0;
:
:
do
   {
       ch = getchar();
       line[c] = ch;
       c++;
   }
while  (ch != '\n');

c = c - 1;
line[c] = '\0';
```

Read characters until CR ('\n') is encountered

Make it a valid string

Reading a line :: Alternate Approach

```
char line[81];
:
:
scanf ("%[ ABCDEFGHIJKLMNOPQRSTUVWXYZ]", line);
```

➔ Reads a string containing uppercase characters and blank spaces

```
char line[81];
:
:
scanf ("%[^\n]", line);
```

➔ Reads a string containing any characters

Writing Strings to the Screen

- We can use printf with the "%s" format specification.

```
char name[50];
:
:
printf ("\n %s", name);
```


Processing Character Strings

- There exists a set of C library functions for character string manipulation.
  - strcpy :: string copy
  - strlen :: string length
  - strcmp :: string comparison
  - strtcat :: string concatenation
- It is required to add the line

```
#include <string.h>
```

strcpy()

- Works very much like a string assignment operator.

```
strcpy (string1, string2);
```

  - Assigns the contents of string2 to string1.
- Examples:

```
strcpy (city, "Calcutta");
strcpy (city, mycity);
```

- Warning:
  - Assignment operator do not work for strings.

```
city = "Calcutta";
```
➔ INVALID

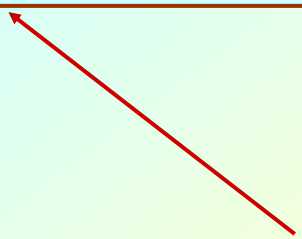
strlen()

- Counts and returns the number of characters in a string.

```
len = strlen (string);
                /* Returns an integer */
```

  - The null character ('\0') at the end is not counted.
  - Counting ends at the first null character.

```
char  city[15];
int  n;
:
:
strcpy (city, "Calcutta");
n = strlen (city);
```



n is assigned 8

strcmp()

- Compares two character strings.

```
int strcmp(string1, string2);
```

  - Compares the two strings and returns 0 if they are identical; non-zero otherwise.
- Examples:

```
if (strcmp(city, "Delhi") == 0)
    {  …….  }

if (strcmp(city1, city2) != 0)
     { ……. }
```

strcat()

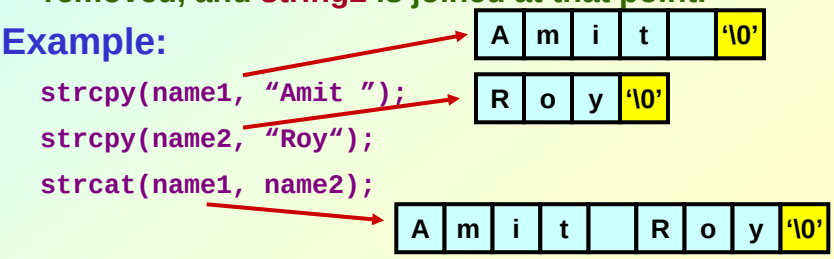
- Joins or concatenates two strings together.

```
strcat (string1, string2);
```

  - string2 is appended to the end of string1.
  - The null character at the end of string1 is removed, and string2 is joined at that point.

- Example:

```
strcpy(name1, "Amit ");
strcpy(name2, "Roy");
strcat(name1, name2);
```



Example

```
/* Read a line of text and count the number of
uppercase letters */
#include  <stdio.h>
#include  <string.h>
main()
{
    char  line[81];
    int  i, n, count=0;
    scanf ("%[^\n]", line);
    n = strlen (line);
    for (i=0; i<n; i++)
       if (isupper(line[i])
             count++;
    printf ("\n The number of uppercase letters in
the string %s is %d", line, count);
}
```

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

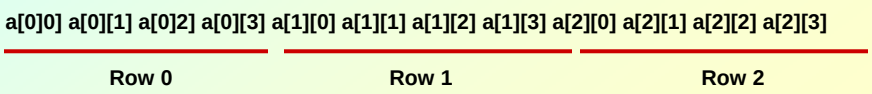
- **Similar to that for 1-D array, but use two indices.**
  - **First indicates row, second indicates column.**
  - **Both the indices should be expressions which evaluate to *integer values*.**
- **Examples:**

```
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

Element `a[i][j]` :: allocated memory location at address x + (i * c + j) * k



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.

- How to print two matrices side by side?

Example: Matrix Addition

```
#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]);
  }
}
```


Passing Arrays 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.

Example: Transpose of a matrix

```
void transpose (x, n)
int x[][3], n;
{
   int  p, q;

   for (p=0; p<n; p++)
     for (q=0; q<n; q++)
      {
         t = x[p][q];
         x[p][q] = x[q][p];
         x[q][p] = t;
       }
}
```

10 20 30

40 50 60

70 80 90

10 20 30

40 50 60

70 80 90

The Correct Version

```
void transpose (x, n)
int x[][3], n;
{
   int  p, q;

   for (p=0; p<n; p++)
     for  (q=p; q<n; q++)
       {
         t = x[p][q];
         x[p][q] = x[q][p];
         x[q][p] = t;
       }
}
```

10 20 30

40 50 60

70 80 90

10 40 70

20 50 80

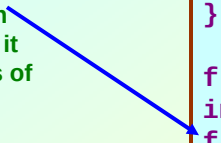
30 60 90

Example Usage

We can also write

float x[100];

But the way the function is written makes it general; it works with arrays of any size.



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

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

Example: Minimum of a set of numbers

```
#include <stdio.h>

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

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

  printf ("\n Minimum is %d",
              minimum(a,n));
}
```

```
int minimum (x,size)
int x[], size;
{
  int i, min = 99999;

  for (i=0;i<size;i++)
  if (min < a[i])
     min = a[i];
  return (min);
}
```

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];

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.