Arrays

CS10001: Programming & Data Structures

Pallab Dasgupta

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?

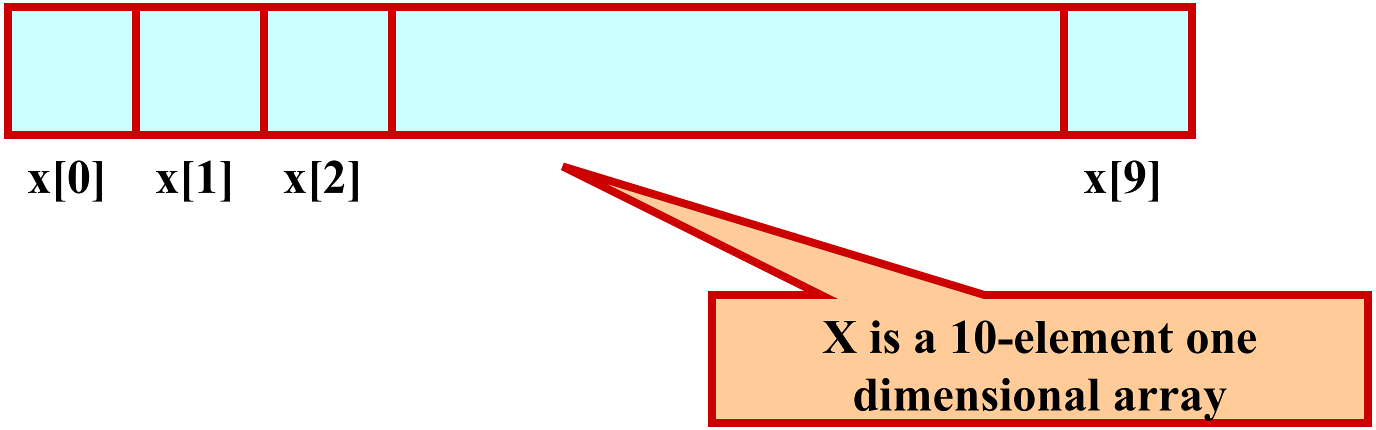
- Solution:
  - Use arrays.

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.

```
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.

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.

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

```
#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

```
#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);
}
```

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

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 Grade point average:  is %d”, gpa);
}
```

Things you can⑦⑧t 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

```
#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) ;
}
```

Contd.

We don’t need to write the array size. 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];
}
```

Arrays 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] ;
}
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];
```

Pointers

Basic Concept

- Within the computer memory, every stored data item occupies one or more contiguous memory cells.
  - The number of memory cells required to store a data item depends on its type (char, int, double, etc.).
- Whenever we declare a variable, the system allocates memory location(s) to hold the value of the variable.
  - Since every byte in memory has a unique address, this location will also have its own (unique) address.

Contd.

- Consider the statement

```
int   xyz = 50;
```

  - This statement instructs the compiler to allocate a location for the integer variable xyz, and put the value 50 in that location.
  - Suppose that the address location chosen is 1380.

xyz → variable

50 → value

1380 → address

Contd.

- During execution of the program, the system always associates the name xyz with the address 1380.
  - The value 50 can be accessed by using either the name xyz or the address 1380.
- Since memory addresses are simply numbers, they can be assigned to some variables which can be stored in memory.
  - Such variables that hold memory addresses are called pointers.
  - Since a pointer is a variable, its value is also stored in some memory location.

Pointers

- A pointer is a variable that represents the location (rather than the value) of a data item.

Contd.

- Suppose we assign the address of xyz to a variable p.
  - p is said to point to the variable xyz.

Variable	Value	Address
xyz	50	1380
p	1380	2545

p = &xyz;

Accessing the Address of a Variable

- The address of a variable can be determined using the ‘&’ operator.
  - The operator ‘&’ immediately preceding a variable returns the address of the variable.
- Example:

  p = &xyz;

  - The address of xyz (1380) is assigned to p.
- The ‘&’ operator can be used only with a simple variable or an array element.

  &distance

  &x[0]

  &x[i-2]

Contd.

- Following usages are illegal:

```
&235
```

  - Pointing at constant.

```
int   arr[20];
  :
&arr;
```

  - Pointing at array name.

```
&(a+b)
```

  - Pointing at expression.

Pointer Declarations

- Pointer variables must be declared before we use them.
- General form:

```
data_type   *pointer_name;
```

Three things are specified in the above declaration:

1. The asterisk (*) tells that the variable pointer_name is a pointer variable.
2. pointer_name needs a memory location.
3. pointer_name points to a variable of type data_type.

Contd.

- Example:

```
int    *count;
float  *speed;
```

- Once a pointer variable has been declared, it can be made to point to a variable using an assignment statement like:

```
int     *p,  xyz;
:
p = &xyz;
```

  - This is called pointer initialization.

Things to Remember

- Pointer variables must always point to a data item of the *same type*.

```
float   x;
int    *p;
:                       ➔ will result in erroneous output
p = &x;
```

- Assigning an absolute address to a pointer variable is prohibited.

```
int   *count;
:
count = 1268;
```

Accessing a Variable Through its Pointer

- Once a pointer has been assigned the address of a variable, the value of the variable can be accessed using the indirection operator (*).

```
int   a, b;
int   *p;
:
p = &a;
b = *p;
```

Equivalent to

b = a

Example 1

```
#include  <stdio.h>
main()
{
   int   a, b;
   int   c = 5;
   int   *p;

   a  =  4  *  (c  +  5) ;

   p  =  &c;
   b  =  4  *  (*p  +  5) ;
   printf  (“a=%d  b=%d \n”,  a, b) ;
}
```

Equivalent

Pointer Expressions

- Like other variables, pointer variables can be used in expressions.
- If p1 and p2 are two pointers, the following statements are valid:

```
sum   =  *p1  +  *p2 ;
prod  =  *p1  *  *p2 ;
prod  =   (*p1)  *  (*p2) ;
*p1  =  *p1  +  2;
x  =  *p1  /  *p2  +  5 ;
```

Pointer Arithmetic

- What are allowed in C?
  - Add an integer to a pointer.
  - Subtract an integer from a pointer.
  - Subtract one pointer from another (related).
    - If p1 and p2 are both pointers to the same array, them p2–p1 gives the number of elements between p1 and p2.
- What are not allowed?
  - Add two pointers.

```
p1  =  p1 + p2 ;
```

  - Multiply / divide a pointer in an expression.

```
p1  =  p2 / 5 ;
p1  =  p1 – p2 * 10 ;
```

Scale Factor

- We have seen that an integer value can be added to or subtracted from a pointer variable.

```
int   *p1,  *p2 ;
int   i,  j;
:
p1  =  p1  +  1 ;
p2  =  p1  +  j ;
p2++ ;
p2  =  p2  –  (i + j) ;
```

- In reality, it is not the integer value which is added/subtracted, but rather the scale factor times the value.

Contd.

Data Type	Scale Factor
char	1
int	4
float	4
double	8

- If p1 is an integer pointer, then

  p1++

  will increment the value of p1 by 4.

Passing Pointers to a Function

- Pointers are often passed to a function as arguments.
  - Allows data items within the calling program to be accessed by the function, altered, and then returned to the calling program in altered form.
  - Called call-by-reference (or by address or by location).
- Normally, arguments are passed to a function by value.
  - The data items are copied to the function.
  - Changes are not reflected in the calling program.

Example: passing arguments by value

```
#include  <stdio.h>
main()
{
     int  a, b;
     a = 5 ;   b = 20 ;
     swap (a, b) ;
     printf  (“\n a = %d,  b = %d”, a, b);
}

void   swap  (int  x, int  y)
{
     int  t ;
     t = x ;
     x = y ;
     y = t ;
}
```

Output

```
a = 5, b = 20
```

Example: passing arguments by reference

```
#include  <stdio.h>
main()
{
     int  a, b;
     a = 5 ;   b = 20 ;
     swap (&a, &b) ;
     printf  (“\n a = %d,  b = %d”, a, b);
}

void   swap  (int  *x, int  *y)
{
     int  t ;
     t = *x ;
     *x = *y ;
     *y = t ;
}
```

Output

a = 20, b = 5

scanf Revisited

```
int   x,  y ;
printf  (“%d %d %d”,  x, y, x+y) ;
```

- What about scanf ?

```
scanf   (“%d %d %d”, x, y, x+y) ;
```

NO

```
scanf   (“%d %d”, &x, &y) ;
```

YES

Example: Sort 3 integers

- Three-step algorithm:
  1. Read in three integers x, y and z
  2. Put smallest in x
     - Swap x, y if necessary; then swap x, z if necessary.
  1. Put second smallest in y
     - Swap y, z if necessary.

Contd.

```
#include  <stdio.h>
main()
{
      int  x, y, z ;
      ………..
      scanf  (“%d %d %d”, &x, &y, &z) ;
      if  (x > y)   swap (&x, &y);
      if  (x > z)   swap (&x, &z);
      if  (y > z)   swap (&y, &z) ;
      ………..
}
```

sort3 as a function

```
#include  <stdio.h>
main()
{
     int  x, y, z ;
     ………..
     scanf  (“%d %d %d”, &x, &y, &z) ;
     sort3  (&x, &y, &z) ;
     ………..
}

void   sort3  (int *xp,  int *yp,  int *zp)
{
     if  (*xp > *yp)   swap (xp, yp);
     if  (*xp > *zp)   swap (xp, zp);
     if  (*yp > *zp)   swap (yp, zp);
}
```

Contd.

- Why no ‘&’ in swap call?
  - Because xp, yp and zp are already pointers that point to the variables that we want to swap.

Pointers and Arrays

- When an array is declared,
  - The compiler allocates a base address and sufficient amount of storage to contain all the elements of the array in contiguous memory locations.
  - The base address is the location of the first element (index 0) of the array.
  - The compiler also defines the array name as a constant pointer to the first element.

Example

- Consider the declaration:

  int x[5] = {1, 2, 3, 4, 5} ;

  – Suppose that the base address of x is 2500, and each integer requires 4 bytes.

Element	Value	Address
x[0]	1	2500
x[1]	2	2504
x[2]	3	2508
x[3]	4	2512
x[4]	5	2516

Contd.

x = &x[0] = 2500 ;

- p = x; and p = &x[0]; are equivalent.
- We can access successive values of x by using p++ or p- - to move from one element to another.

- **Relationship between p and x:**

p	=	&x[0]	=	2500
p+1	=	&x[1]	=	2504
p+2	=	&x[2]	=	2508
p+3	=	&x[3]	=	2512
p+4	=	&x[4]	=	2516

*(p+i) gives the

value of x[i]

Example: function to find average

```
#include  <stdio.h>
main()
{
     int  x[100], k, n ;

     scanf  (“%d”, &n) ;

     for  (k=0; k<n; k++)
         scanf  (“%d”, &x[k]) ;

     printf  (“\nAverage is %f”,
                                 avg (x, n));
}
```

```
float  avg  (array, size)
int  array[], size ;
{
     int  *p, i , sum = 0;

     p = array ;

     for  (i=0; i<size; i++)
          sum = sum + *(p+i);

     return  ((float) sum / size);
}
```