CS - 114 : Computer Workshop

Prof. Chamakuri Nagaiah
Mahindra-École Centrale, Hyderabad
nagaiah.chamakuri@mechyd.ac.in

Introduction to Arrays

- Many applications require multiple data items that have common characteristics.
- Finding of an Avg of second year students marks, etc.....

Introduction to Arrays

- Many applications require multiple data items that have common characteristics.
- Finding of an Avg of second year students marks, etc.....

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;
```

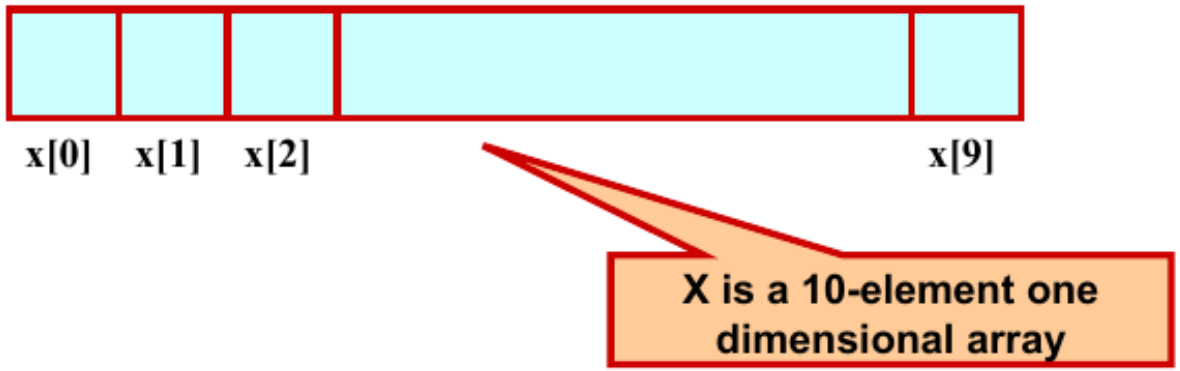
The problem

- Suppose we have 10 numbers to handle.
- Or 20.
- Or 20 Or 100 or 10000.
- Where do we store the numbers ? Use 10000 variables ??
- How to tackle this problem?
- Solution:
  - Use arrays.
- One dimensional and multidimentional arrays

Using Arrays

- An array is a collection of data that holds fixed number of values of same type (float/int/char)
- All the data items constituting the group share the same name.

  int x[10];
- All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



- Arrays have 0 as the first index not 1. In this example, x[0]

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[20];
  - marks is an array containing a maximum of 20 integers.

Declaring Arrays

- Examples:

  int x[10];

  char line[80]; // character array i.e. string

  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.

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.
- 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;
```

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[3] = {'M', 'E', 'C'};

- 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};

  $\rightarrow$ total[0]=24.2, total[1]=-12.5, total[2]=35.1, total[3]=0, total[4]=0

Initialization of Arrays

- General form:

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

- Examples:

  int marks[5] = {72, 83, 65, 80, 76};

  char name[3] = {'M', 'E', 'C'};

- 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

  – 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[ ] = {'C', 'S', 'i', 't'};

Character Arrays and Strings

- char C[8] = { 'c','o','m','p','u','t','e','r','\0' }
- C[0] gets the value 'c', C[1] the value 'o', and so on. The last 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] = "computer";
- 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: Accessing Array Elements

```
#include <stdio.h>
 int main () {
 int n[ 10 ]; /* n is an array of 10 integers */
 int i,j;

 /* initialize elements of array n to 0 */
 for ( i = 0; i < 10; i++ ) {
    n[i] = i+100; /*set element at location i to i+100 */
 }

 /* output each array element's value */
 for (j = 0; j < 10; j++ ) {
    printf("Element[%d] = %d\n", j, n[j] );
 }
  return 0;
}
```

Example 1: Output

```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

Example 2: 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);
}
```

Things you can't do

- 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.

Multi-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

- 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 supports multidimensional arrays. The simplest form of the multidimensional array is the two-dimensional array.

Declaring 2-D Arrays

- A two-dimensional array is, in essence, a list of one-dimensional arrays.
- General form:

Declaring 2-D Arrays

- A two-dimensional array is, in essence, a list of one-dimensional arrays.
- General form:

  type array_name [row_size][column_size];

  Where **type** can be any valid C data type and **array_name** will be a valid C identifier.

Declaring 2-D Arrays

- A two-dimensional array is, in essence, a list of one-dimensional arrays.
- General form:

  type array_name [row_size][column_size];

  Where **type** can be any valid C data type and **array_name** will be a valid C identifier.
- Examples:

  int marks[4][5];

Declaring 2-D Arrays

- A two-dimensional array is, in essence, a list of one-dimensional arrays.
- General form:

  type array_name [row_size][column_size];

  Where **type** can be any valid C data type and **array_name** will be a valid C identifier.
- Examples:

  int marks[4][5];

  float sales[12][25];

Declaring 2-D Arrays

- A two-dimensional array is, in essence, a list of one-dimensional arrays.
- General form:

  type array_name [row_size][column_size];

  Where **type** can be any valid C data type and **array_name** will be a valid C identifier.
- Examples:

  int marks[4][5];

  float sales[12][25];

  double matrix[100][100];

Declaring 2-D Arrays

- A two-dimensional array is, in essence, a list of one-dimensional arrays.
- General form:

  type array_name [row_size][column_size];

  Where **type** can be any valid C data type and **array_name** will be a valid C identifier.
- Examples:

  int marks[4][5];

  float sales[12][25];

  double matrix[100][100];
- multidimensional arrays
  - General form : type name[size1][size2]...[sizeN];

Declaring 2-D Arrays

- A two-dimensional array is, in essence, a list of one-dimensional arrays.
- General form:

  type array_name [row_size][column_size];

  Where **type** can be any valid C data type and **array_name** will be a valid C identifier.
- Examples:

  int marks[4][5];

  float sales[12][25];

  double matrix[100][100];
- multidimensional arrays
  - General form : type name[size1][size2]...[sizeN];
  - Example : int threedim[5][10][4];

Declaring 2-D Arrays

- 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.

Declaring 2-D Arrays

- 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.
- A two-dimensional array **a**, which contains three rows and four columns can be shown as follows

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

- Thus, every element in the **array a** is identified by an element name of the form **a[i][j]**, where 'a' is the name of the array, and 'i' and 'j' are the subscripts that uniquely identify each element in 'a'.

Initializing Two-Dimensional Arrays

- Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row has 4 columns.

```
int a[3][4] = {
 {0, 1, 2, 3} , /* initializers for row indexed by 0 */
 {4, 5, 6, 7} , /* initializers for row indexed by 1 */
 {8, 9, 10, 11} /* initializers for row indexed by 2 */
};
```

Initializing Two-Dimensional Arrays

- Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row has 4 columns.

```
int a[3][4] = {
 {0, 1, 2, 3} , /* initializers for row indexed by 0 */
 {4, 5, 6, 7} , /* initializers for row indexed by 1 */
 {8, 9, 10, 11} /* initializers for row indexed by 2 */
};
```

- The nested braces, which indicate the intended row, are optional.
- The following initialization is equivalent to the above example

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

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.
- Printing the elements of a 2-D array??

Example 1 : Accessing 2-D Array Elements

```
  #include <stdio.h>
int main () {
   /* an array with 5 rows and 2 columns*/
   int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
   int i, j;

   /* output each array element's value */
   for ( i = 0; i < 5; i++ )
      for ( j = 0; j < 2; j++ )
         printf("a[%d][%d] = %d\n", i,j, a[i][j] );

   return 0;
}
```

Example 2: 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 functions

- Array element can be passed to functions as an ordinary arguments.

  IsFactor (x[i], x[0])

  sin (x[5])

Passing arrays to functions

- Array element can be passed to functions as an ordinary arguments.

  IsFactor (x[i], x[0])

  sin (x[5])

- 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.

Passing arrays to functions

- Array element can be passed to functions as an ordinary arguments.

  IsFactor (x[i], x[0])

  sin (x[5])

- 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.

Passing arrays to functions

- Passing a **1D array** as an argument in a function, you would have to declare a formal parameter in one of following 3 ways
  - Way-1 : Formal parameters as a pointer

```
void myFunction(int *param) {
    .....
}
```

Passing arrays to functions

- Passing a **1D array** as an argument in a function, you would have to declare a formal parameter in one of following 3 ways
  - Way-1 : Formal parameters as a pointer

```
void myFunction(int *param) {
    .....
}
```

  - Way-2 : Formal parameters as a sized array

```
void myFunction(int param[10]) {
    .....
}
```

Passing arrays to functions

- Passing a **1D array** as an argument in a function, you would have to declare a formal parameter in one of following 3 ways
  - Way-1 : Formal parameters as a pointer

```
void myFunction(int *param) {
    .....
}
```

  - Way-2 : Formal parameters as a sized array

```
void myFunction(int param[10]) {
    .....
}
```

  - Way-3 : Formal parameters as an unsized array

```
void myFunction(int param[]) {
    .....
}
```

- Similarly, you can pass multi-D arrays as formal parameters.

Example 1 : Whole array as Parameters

```
double getAverage(int arr[], int size) {

 int i;
 double avg;
 double sum = 0;
 /* calculate sum of elements */
 for (i = 0; i < size; ++i) {
    sum += arr[i];
 }

 avg = sum / size;
/* return avg as double */
 return avg;
}
```

Example 1 : Whole array as Parameters

```
#include <stdio.h>
/* function declaration */
double getAverage(int arr[], int size);

int main () {
   /* an int array with 5 elements */
   int balance[5] = {1000, 2, 3, 17, 50};
   double avg;

   /* pass pointer to the array as an argument */
   avg = getAverage( balance, 5 ) ;

   /* output the returned value */
   printf( "Average value is: %f ", avg );
   return 0;
}
```

Example 2: 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] ;
 }

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}, 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.

The Actual Mechanism

- 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 : Passing 2-D Arrays

```
#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];
```

Return array from function in C

- First point: C programming does not allow to return an entire array as an argument to a function.
- However, you can return a pointer to an array by specifying the array's name without an index.
- Example : return a single-dimension array from a function

```
int * myFunction() {
    .....
  }
```

- Second point to remember is that C does not advocate to return the address of a local variable to outside of the function, so you would have to define the local variable as static variable.

Example

```
/* function to generate and return random numbers */
int * getRandom( ) {

   static int  r[10];
   int i;

   /* set the seed */
   srand( (unsigned)time( NULL ) );

   for ( i = 0; i < 10; ++i) {
      r[i] = rand();
      printf( "r[%d] = %d\n", i, r[i]);
   }
   /* return pointer r */
   return r;
}
```

Example : Contd.

```
#include <stdio.h>
/* main function to call above defined function */
int main () {

   /* a pointer to an int */
   int *p;
   int i;
   /* get a pointer */
   p = getRandom();

   for ( i = 0; i < 10; i++ ) {
      printf( "*(p + %d) : %d\n", i, *(p + i));
   }

   return 0;
}
```