Arrays

Chapter 06

CMPE-112 *Programming Fundamentals*

Lecture Plan

- Two sample programs
- Basics of Arrays
  - Dimensions
  - Array Declaration
  - Accessing Array Elements
  - Copying
  - Array Initialization
- Arrays as Function Arguments
  - Passing Array Elements as Arguments
  - Passing Arrays as Arguments
- Sample Program

Sample Program (I)

```
/* The program computes the product and          */
/*   the sum of n numbers without using arrays  */
#include  <stdio.h>

double sum( int a )
{
  int i;
  double s = 0.0, num;

  for (i=1; i<=a; i++)
  {
    printf("\nEnter the value #%d - ", i);
    scanf("%lf", &num);
    s += num;
  }

  return s;
}
```

```
double product( int a )
{
  int i;
  double prod = 1.0, num;

  for (i=1; i<=a; i++)
  {
    printf("\nEnter the value #%d - ", i);
    scanf("%lf", &num);
    prod *= num;
  }

  return prod;
}

int main()
{
  int items;

  puts("\nEnter the number of factors: ");
  scanf("%d", &items);
  printf("\nThe product = %.2lf\n", product(items));
  printf("\nThe sum     = %.2lf\n", sum(items));

  return 0;
}
```

3

3

Sample Program (II)

```
/* The program computes the product and   */
/*    the sum of n numbers using arrays       */
#include  <stdio.h>

#define  MAX_EL  10

double sum( double ar[], int a )
{
  int i;
  double s = 0.0, num;

  for (i=0; i<a; i++)
      s += ar[i];

  return s;
}

double product( double ar[], int a )
{
  int i;
  double prod = 1.0, num;

  for (i=0; i<a; i++)
      prod *= ar[i];

  return prod;
}
```

```
void assign( double m[], int el );   // This is a prototype

int main()
{
  int items;
  double f_array[MAX_EL];

  puts("\nEnter the number of factors: ");
  scanf("%d", &items);
  if (items <= MAX_EL)  {
    assign(f_array, items);
    printf("\nThe product = %.2lf\n", product(f_array, items));
    printf("\nThe sum     = %.2lf\n", sum(f_array, items));
  }
  else
    puts("\nWarning: you entered too big value!");

  return 0;
}

void assign( double vect[], int elements )
{
  int i;

  for (i=0; i<elements; i++)  {
    printf("\nEnter the value #%d - ", i+1);
    scanf("%lf", &vect[i]);
  }
}
```

4

4

Basics of Arrays

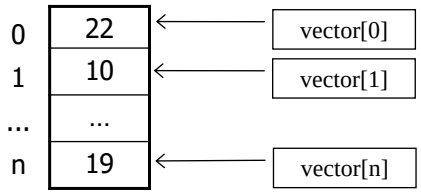
- An ordered finite collection of data items, each of the same type, is called an *array*, and the individual data items are its *elements*
- *Array_Name* is assigned to an entire array, and individual elements are referenced by specifying a *subscript* (also called an *index*)
- An array has the following properties:
  - the *type* of an array is the data type of its elements
  - the *location* of an array is the location of its first element
  - the *length* of an array is the number of data elements in the array
  - the *size* of an array is the length of the array times the sizes of an element

5

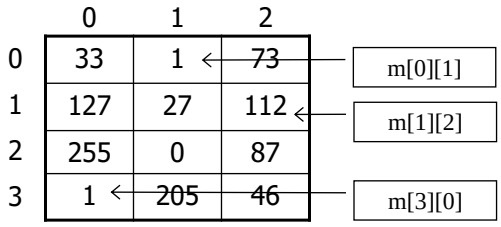
5

Dimensions

- Arrays with **one** subscript are called *linear* or *one-dimensional* arrays



- Arrays with **two** subscripts are called *two-dimensional* arrays, etc.



6

6

Array Declaration

- Arrays must be declared before use. An array declaration is of the form:

  type array_name[expr_1][expr_2]…[expr_n];

  char student_id[20][6];

- Any constant integral expression may be used to specify the number of elements in an array, for example

  #define ROWS 20

  #define COLUMNS 10

  static double table[ROWS][COLUMNS];

  *auto double values[ROWS * COLUMNS];*

- Arrays can be of automatic or static storage class, so the type can be prefixed with *auto* or *static* keywords

7

7

Accessing Array Elements

- An element of an array can be accessed by specifying appropriate subscripts with the array name. *Any integral expression* (not just constant) can be used as a subscript:

  *table[5][i*2] = table[k/2][i] + table[k/2][j];*

  *++values[3*ROWS+7];*

- Subscripts should not have values outside the array bounds. **C does not automatically check subscripts to lie within the array bounds** - this must be the programmer's concern

- An array element can be used anywhere that a simple variable can be used

  printf("This is one array element: %lf", table[i][j]);

8

8

Copying

- An array can NOT be copied into another array by assigning it to that array
- For example, two arrays are declared as follows

```
int  from[10];
int  to[10];
```

So, the assignment statements such as

```
to = from;
to[] = from[];
```

are illegal. **The correct way is to copy them element-by-element** using an appropriate loop:

```
for(i=0; i<10; i++)
   to[ i ] = from[ i ];
```

9

Array Initialization (I)

- Elements of an array can be assigned initial values at the declaration of the array using a list of initializers enclosed in braces and separated by commas, for example

```
int  score[5] = { 41, 97, 10, 55, 100 };
```

- Multi-dimensional arrays are initialized in the same manner:

```
int  books[3][4] =
   {
     {  11,   32,   21, 250 },
     { 211,    7, 162,    2 },
     {  15, 180,   48, 190 }
   };
```

```
int  books[3][4] =
    {  11, 32, 21, 250, 211, 7,
       162, 2, 15, 180, 48, 190 };
```

10

Array Initialization (II)

- Some specific rules govern array initialization:

1. If the number of initializers is less than the number of elements in the array, the remaining elements are set to zero:

   int a[5] = { 3, 2, 5 }; ≡ *int a[5] = { 3, 2, 5, 0, 0 };*

2. If initializers are provided for an array, the array length may be omitted:

   int b[] = { 7, 11, 100, 4 }; ≡ *int b[4] = { 7, 11, 100, 4 };*

3. A character array me be initialized by a string constant:

   char os[] = "UNIX"; ≡ *char os[5] = { 'U', 'N', 'I', 'X', '\0' };*

   char comp[4] = "Ltd"; ≡ *char comp[4] = { 'L', 't', 'd', '\0' };*

11

11

Passing Array Elements as Arguments

- To pass an array element to a function, the array element is specified in the function call just as a simple variable is specified:

  double x, arr[10];

  x = sin(arr[2]); // Finds a sine of the third element

- Individual elements are passed **by value**
- If the type of the array element is different from the argument type expected by the function, conversion rules are applied

12

12

Passing Arrays as Arguments

- Functions can be defined to take arrays as arguments:

```
void square( double arr[][3], int a )
{
  int i, j;

  for (i=1; i<=a; i++)
      for(j=1; j<=3; j++)
          arr[i][j] = arr[i][j] * arr[i][j];
}
```

```
int main(void)
{
  double  x[7][3];
  double  y[4][3];

  square(x, 7);      // A function call
  square(y, 4);      // A function call

  return 0;
}
```

- Arrays are passed to a function **by reference**; when an array is passed as an actual argument, *the address* of the beginning of the array is passed and the array elements are not copied into the formal parameter array

13

Sample Program (III)

```
/* This program enters and     */
/*   concatenates two strings  */
#include  <stdio.h>

#define  MAX_LEN  80

void concatenate( char s1[], char s2[], char s_res[] )
{
  int i, j;

  i=0;
  j=0;
  while(s1[j] != '\0') {          //   Copy the first string
    s_res[i] = s1[j];
    i++; j++;
  }

  for(j=0; s2[j] != '\0'; j++) {  //   Copy the second string
    s_res[i] = s2[j];
    i++;
  }
  s_res[i] = '\0';
}
```

```
int main()
{
  int n;
  char str1[MAX_LEN],
       str2[MAX_LEN],
       str_result[MAX_LEN];

  puts("\nEnter the first string:");
  gets(str1);
  puts("\nEnter the second string:");
  gets(str2);

  concatenate(str1, str2, str_result);

  puts("\nThe resulting string:");
  puts(str_result);

  return 0;
}
```


14