Chapter 6

Array and Strings

Objective

- Understand what an array is
- Learn about one-dimensional array, their declaration, initialization, ways to access individual array elements, representation of array elements in memory, and other possible operations
- Learn about one-dimensional strings and the way they are declared, initialized, manipulated, inputted, and displayed
- Learn about two-dimensional arrays, initialization of sized and unsized two-dimensional arrays, accessing elements in such arrays, and how this kind of an array can be used
- Know about array of strings, its declaration, initialization, other operations, manipulations, and uses
- Get a brief idea of three-dimensional arrays or even larger ones

Introduction

- *A scalar variable is* a single variable whose stored value is an atomic type.
  - For example, each of the variables ch, n, and price declared in the statements

    char ch;

    int n;

    float price;
  - are of different data types and each variable can only store one value of the declared data type. These types of variables are called *scalar variables.*
  - This means that the value cannot be further subdivided or separated into a legitimate data type.
  - An aggregate type, which is referred to as both a *structured type* and a *data structure,* is any type whose values can be decomposed and are related by some defined structure.

Key Words

- **Aggregate data type:** It is an agglomeration of data, of any data type, that is identified with a single name and can be decomposed and related by some defined structure.
- **Array identifier:** A name assigned to an array.
- **Array initialization:** The procedure of assigning numerical value or character to each element of an array.
- **Array of strings:** An array that contains strings as its elements.
- **Array:** It is a collection of individual data elements that is ordered, fixed in size, and homogeneous.
- **Concatenation of strings:** A kind of string manipulation where one string is appended to another string.

Key Words

- **Homogeneous data:** Data of same kind or same data type.
- **Index of an array:** It is an integer constant or variable ranging from 0 to (size – 1).
- **Library functions:** Pre-written functions, provided with the C compiler, which can be attached to user written programs to carry out some task.
- **Multi-dimensional array:** An array that is represented by a name and more than one index or subscript.
- **One-dimensional array:** An array that is represented by a name and single index or subscript.
- **Scalar variable:** It is a single variable whose stored value is an atomic data type.
- **Scanset:** It is a conversion specifier that allows the programmer to specify the set of characters that are (or are not) acceptable as part of the string.

Key Words

- **Size of array:** The number of elements in an array.
- **Stderr:** The side stream of output characters for errors is called standard-error.
- **Stdin:** Standard input stream that is used to receive and hold input data from standard input device.
- **Stdout:** Standard output stream that is used to hold and transfer output data to standard output device.
- **String comparison:** A kind of string manipulation where two strings are compared to primarily find out whether they are same or not.
- **String copy:** A kind of string manipulation where one string is copied into another.
- **String manipulation:** Carrying out various operations like comparing, appending, copying, etc. between strings.
- **String :** One-dimensional array of characters that contain a NUL at the end.

Why Array?

- Consider a brand-new problem: a program that can print its input in reverse order. If there are two or three values, this is easy but what if there are 10 or 20 or 100 values? Then it is not so easy. To avoid above problem we use array .
  - A scalar variable is a single variable whose stored value is an atomic data type.
  - An array is a collection of individual data elements that is ordered, fixed in size, and homogeneous.
  - An array is considered to be a derived data type.
  - Array enables the storing and manipulation of potentially huge quantities of data.

One-dimensional Array

- There are several forms of an array used in C:
  - One-dimensional or single-dimensional and multidimensional array
  - Since the array is one dimensional, there will be a single subscript or index whose value refers to the individual array element which ranges from 0 to (*n*–1), where *n* is the total number of elements in the array.
- When defining an array in a program three things need to be specified.
  - The type of data it can hold, i.e., int, char, double, float, etc.
  - The number of values it can hold, i.e., the maximum number of elements it can hold
  - A name

One-dimensional Array

- The array *size* must be a positive integer number or an expression that evaluates to a positive integer number that must be specified at the time of declaration with the exception that it may be unspecified while initializing the array.
- In C, the array index starts at 0 and ends at (size–1) and provides the means of accessing and modifying the specific values in the array.
- C never checks whether the array index is valid—either at compile time or at run time.

Syntax: Array

- The syntax for declaration of a one-dimensional array is data_type array_name [SIZE];
  - All the array elements hold values of type <data type>
  - The size of the array is indicated by <SIZE>, the number of elements in the array. <SIZE> must be an int constant or a constant expression.
- int a[size]; /* memory space for a[0],a[1],…, a[size –1] allocated */

  lower bound = 0

  upper bound = size –1

  size = upper bound + 1

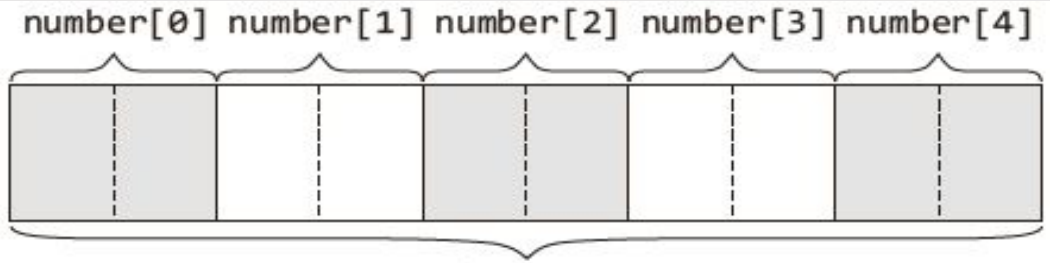


Figure Identifying individual array elements

  - number[0] & number[4] refer to the first and fifth numbers stored in the ‘number’ array respectively .

Initializing Integer Arrays

- Variables can be assigned values during declaration, like in the following example.

  int x = 7;

- Array initialization statements are as shown.

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

  9 8 7 6 5 4 3 2 1 0 ----- values stored in array elements

  0 1 2 3 4 5 6 7 8 9-------index values of array elements

  (b) double a[5] = {3.67, 1.21, 5.87, 7.45, 9.12}

- Automatic sizing while initializing, the size of a one dimensional array can be omitted as shown.

  int arr[] = {3,1,5,7,9};

  Here, the C compiler will deduce the size of the array from the initialization statement.

Accessing Array Elements

- x and y are similar arrays (i.e., of the same data type, dimensionality, and size), then assignment operations, comparison operations, etc., involving these two arrays must be carried out on an element-by-element basis.
- Examples using the elements of an array named ‘numbers’ are shown here:

```
numbers [0] = 98;
numbers [1] = numbers [0] – 11
numbers [2] = 2 * (numbers [0] – 6);
numbers [3] = 79;
numbers [4] = (numbers [2] + numbers [3] – 3)/2;
total = numbers[0] + numbers[1] + numbers[2] + numbers[3] + numbers[4];
```

Accessing Array Elements

- This makes statements such as:

  - total = numbers[0] + numbers[1] + numbers[2] + numbers[3] + numbers [4];

- One extremely important advantage of using integer expressions as subscripts is that it allows sequencing through an array using a for loop.

- Example

```
total = 0; /*initialize total to zero */
for(i = 0; i <5; ++i)
    total = total + numbers[i]; /* add in a number */
```

ARRAY: Other Allowed Operations

- These operations include the following
  - (a) To increment the *i*-th element, the given statements can be used.

```
ar[i]++;
ar[i] += 1;
ar[i] = ar[i] + 1;
```

  - (b) To add n to the *i*-th element, the following statements may be used,

```
ar[i] += n;
ar[i] = ar[i] + n;
```

  - (c) To copy the contents of the *i*-th element to the *k*-th element, the following statement may be written.

```
ar[k] = ar[i];
```

  - (d) To copy the contents of one array ‘ar’ to another array ‘br’, it must again be done one by one.

```
int ar[10], br[10];
for(i = 0; i < 10; i = i + 1)
br[i] = ar[i];
```

  - (e) To exchange the values of ar[i] and ar[k], a ‘temporary’ variable must be declared to hold one value, and it should be of the same data type as the array element being swapped.

```
int temp;
temp = ar[i];
/* save a copy of value in ar[i] */
ar[i] = ar[j];
/* copy value from ar[j] to ar[i] */
ar[j] = temp;
/* copy saved value of ar[i] to ar[j]
   */
```

Storing Values Given by the User in an Array

- Reading the input into an array is done as shown.

```
int a[10]; /* an array with 10 “int” elements */
 int i;
 for(i=0 ; i< 10; i++)
 scanf(“%d”, &a[i]);
```

- The idea is that first a value must be read and copied into a[0], then another value read and copied into a[1], and so on, until all the input values have been read.

Printing an Array

- The following code segment prints the elements of an array, a[10].

```
for(i=0 ; i< 10; i++)
printf("%d", a[i]);
```

- For printing of numbers entered by the user in the reverse order, the program will be as shown

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int a[30],n,i;
/* n = number of array elements and i=index */
printf("\n Enter the number n");
scanf("%d",&n);
if(n>30)
{
  printf("\n Too many Numbers");
  exit(0);
}
for(i=0 ; i< n; i++)
  scanf("%d", &a[i]);
  printf("\n Numbers entered in reverse order \n");
for(i=n-1 ; i>=0; i--)
  printf("%d", a[i]);
  return 0;
}
```

Internal Representation of Arrays in C

- ***References to elements outside of the array bounds:*** It is important to understand that there is no array bound checking in C.
- ***A bit of memory allocation:*** It has been seen how arrays can be defined and manipulated. It is important to learn how to do this because in more advanced C programs it is necessary to deal with something known as dynamic memory management.
  - It is given a small area of the computer's memory to use. This memory, which is known as the *stack,* is used by variables in the program

```
int a = 10;
float values[100];
```

Variable Length Arrays and the C99 Changes

- With an earlier version of C(C89) compilers, an array's size must be a *constant integral expression* so that it can be calculated at compile-time.
  - This has already been mentioned in earlier sections.
- But in the C99 compilers, an array size can be an *integral expression* and not necessarily a constant one.
  - This allows the programmer to declare a *variable-length array* or an array whose size is determined at runtime.
- In the next slide, a program illustrates this concept:
  - Some changes in initializing an array has been made in C99.
  - In C99, initial values can be set only for certain elements, with uninitialized elements being initialized as 0.

Following Program Illustrates the Concept

```
#include <stdio.h>
int main(void)
{
   int n,i;
   printf("\n enter the value of n: ");
   scanf("%d", &n);
   int a[n];
   printf("\n enter the values one by one\n");
   for(i=0;i<n; ++i)
     scanf("%d", &a[i]);
   printf("\n entered numbers are.....\n");
   for(i=0;i<n;++i)
     printf("\n %d",a[i]);
   return 0;
}
```

Application : One-Dimensional Array

- ***Printing binary equivalent of a decimal number using array:*** Here the remainders of the integer division of a decimal number by 2 are stored as consecutive array elements.
  - The division procedure is repeated until the number becomes 0.

```
#include <stdio.h>
int main()
{
  int a[20],i,m,n,r;
  printf("\n Enter the decimal Integer");
  scanf("%d",&n);
  m=n;
  for(i=0;n>0;i++)
  {
    r=n%2;
    a[i]=r;
    n=n/2;
  }
  printf("\n Binary equivalent of %d is \t",m);
  for(i--;i>=0;i--)
  printf("%d",a[i]);
  return 0;
}
```

Fibonacci Series using an Array

```
#include <stdio.h>
int main()
{
   int fib[15];
   int i;
   fib[0] = 0;
   fib[1] = 1;
   for(i = 2; i < 15; i++)
   fib[i] = fib[i-1] + fib[i-2];
   for(i = 0; i < 15; i++)
   printf("%d\n", fib[i]);
   return 0;
}
```

Output:

```
0
1
1
2
3
5
8
13
21
34
55
89
144
233
377
```

Searching an Element within an Array

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
 int a[30],n,i,key, FOUND=0;
 printf("\n How many numbers");
 scanf("%d",&n);
 if(n>30)
 {
  printf("\n Too many Numbers");
  exit(0);
 }
 printf("\n Enter the array elements \n");
 for(i=0 ; i<n; i++)
 scanf("%d", &a[i]);
 printf("\n Enter the key to be searched \n");
 scanf("%d",&key);
 for(i=0 ; i<n; i++)
```

```
if(a[i] == key)
{
  printf("\n Found at %d",i);
  FOUND=1;
}
if(FOUND = = 0)
printf("\n NOT FOUND...");
return 0;
}
```

Sorting an Array

Bubble Sort

- Bubble sort compares adjacent array elements and exchanges their values if they are out of order.
- In this way, the smaller values ‘bubble’ to the top of the array (towards element 0), while the larger values sink to the bottom of the array.
- This sort continues until no exchanges are performed in a pass.

	Pass 1	Pass 2	Pass 3	Pass 4	Pass 5
42	42	26	26	26	26
60	26	42	34	28	28
26	55	34	28	34	34
55	34	28	42	42	42
34	28	55	55	55	55
28	60	60	60	60	60

C Implementation of Bubble Sort

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int a[30],n,i,j,temp, sorted=0;
  printf("\n How many numbers");
  scanf("%d",&n);
  if(n>30)
  {
    printf("\n Too many Numbers");
    exit(0);
  }
  printf("\n Enter the array elements \n");
  for(i=0 ; i< n; i++)
  scanf("%d", &a[i]);
  for(i = 0; i < n-1 && sorted==0; i++)
{
  sorted=1;
  for(j = 0; j < (n - i) -1; j++)
  if(a[j] > a[j+1])
  {
    temp = a[j];
    a[j] = a[j+1];
    a[j+1] = temp;
    sorted=0;
  }
}
printf("\n The numbers in sorted order \n");
for(i=0 ; i<n; ++i)
printf("\n %d", a[i]);
return 0;
}
```

Binary Searching

- Consider the array 1 2 3 4 5 6 7 8 9
- Construct the binary search algorithm for finding the key = 7.
- **1st iteration**
  - HIGH = 8, LOW = 0; because the array index begins with '0' and ends with '8'.

MID = 4, Array[4] = 5, 5<7 : TRUE
LOW = 5
New List = 6 7 8 9

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Data	23	27	29	32	34	41	46	47	49	52	55	68	71	74	77	78
1st iteration	L						M									H
2nd iteration								L				M				H
3rd iteration								L	M	H						

Depiction of binary search algorithm

Binary Searching

- **2nd iteration**
  - HIGH = 8, LOW = 5
    MID = 6, Array[6] = 7, 7<7 : FALSE
    HIGH = 6
    New List = 6 7
- **3rd iteration**
  - HIGH = 6, LOW = 5
    MID = 5, Array[5] = 6, 6<7 : TRUE
    LOW = 6
    New List = 7
- **4th iteration**
  - HIGH = 6, LOW = 6
    MID = 6, Array [MID] = Array [6] = 7 == Key then
    Found = TRUE

Note

- Single operations, which involve entire arrays, are not permitted in C.
- Neither can all elements of an array be set at once nor can one array be assigned to another.
- For an array of length L and data type X, the compiler allocates L* sizeof (X) bytes of contiguous space in memory.

Strings: One-dimensional Character Arrays

- Strings in C are represented by arrays of characters. The end of the string is marked with a special character, the *null character*, which is a character all of whose bits are zero, i.e., a NUL (not a NULL).
  - The null character has no relation except in name to the *null pointer.*
  - In the ASCII character set, the null character is named NUL. The null or string-terminating character is represented by another character escape sequence, \0.

Declaration of a String

- Strings can be declared like one-dimensional arrays.
  - For example,

    char str[30];

    char text[80];
- An array formed by characters is a string in C.
- The end of the string is marked with the null character.
- When the character array size is explicitly specified and the number of initializers completely fills the array size, the null character is not automatically appended to the array.

Printing Strings

- The conversion type ‘s’ may be used for output of strings using printf().
- The following points should be noted.
  - When the field width is less than the length of the string, the entire string is printed.
  - The integer value on the right side of the decimal point specifies the number of characters to be printed.
  - When the number of characters to be printed is specified as zero, nothing is printed.
  - The minus sign in the specification causes the string to be printed as left justified.

Example

```
#include <stdio.h>
int main()
{
char s[]=“Hello, World”;
printf(“>>%s<<\n”,s);
printf(“>>%20s<<\n”,s);
printf(“>>%-20s<<\n”,s);
printf(“>>%.4s<<\n”,s);
printf(“>>%-20.4s<<\n”,s);
printf(“>>%20.4s<<\n”,s);
return 0;
}
```

The output

```
>>Hello, World<<
>> Hello, World<<
>>Hello, World <<
>>Hell<<
>>Hell <<
>> Hell<<
```

String Input/Output

- One special case, where the null character is not automatically appended to the array, is when the array size is explicitly specified and the number of initializers completely fills the array size.
- printf() with the width and precision modifiers in the %s conversion specifier may be used to display a string.
- The %s format does not require the ampersand before the string name in scanf().

String Input/Output

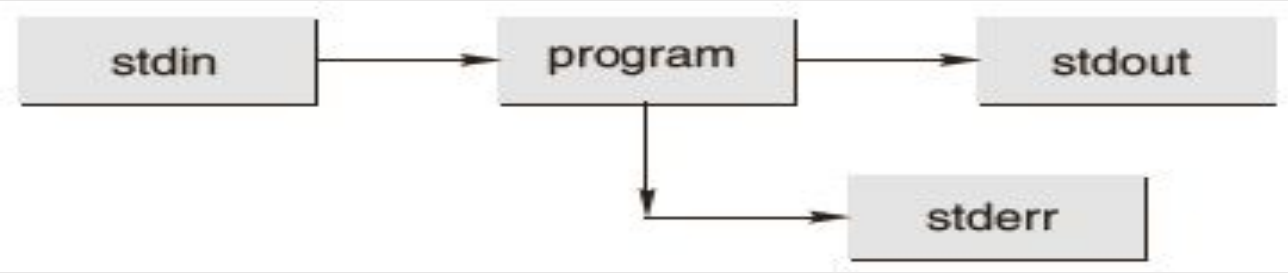
- If fewer input characters are provided, scanf() hangs until it gets enough input characters.
- scanf() only recognizes a sequence of characters delimited by white space characters as an external string.
- While using scanset with scanf(), dangling characters must be ‘absorbed away’ by a subsequent call to scanf() with %c or to getchar().
- The library function sprintf() is similar to printf().
  - The only difference is that the formatted output is written to a memory area rather than directly to a standard output.

Character Manipulation in Strings

Function	Description
`ialnum(c)`	Returns a non-zero if c is alphabetic or numeric
`isalpha(c)`	Returns a non-zero if c is alphabetic
`scntrl(c)`	Returns a non-zero if c is a control character
`isdigit(c)`	Returns a non-zero if c is a digit, 0 – 9
`isgraph(c)`	Returns a non-zero if c is a non-blank but printing character
`islower(c)`	Returns a non-zero if c is a lowercase alphabetic character, i.e., a – z
`isprint(c)`	Returns a non-zero if c is printable, non-blanks and white space included
`ispunct(c)`	Returns a non-zero if c is a printable character, but not alpha, numeric, or blank
`isspace(c)`	Returns a non-zero for blanks and these escape sequences: '\f', '\n', '\r', '\t', and '\v'
`isupper(c)`	Returns a non-zero if c is a capital letter, i.e., A – Z
`isxdigit(c)`	Returns a non-zero if c is a hexadecimal character: 0 – 9, a – f, or A – F
`tolower(c)`	Returns the lowercase version if c is a capital letter; otherwise returns c
`toupper(c)`	Returns the capital letter version if c is a lowercase character; otherwise returns c

String Input and Output using fscanf() and fprintf()

- stdin, stdout, and stderr: Each C program has three I/O streams.
  - The input stream is called standard-input (stdin); the output stream is called standard-output (stdout); and the side stream of output characters for errors is called standard error (stderr).
  - Internally they occupy file descriptors 0, 1, and 2 respectively.
  - Now one might think that calls to fprinf() and fscanf() differ significantly from calls to printf() and scanf().
  - fprintf() sends formatted output to a stream and fscanf() scans and formats input from a stream.



See the Following Example

```
#include <stdio.h>
int main()
{
int first, second;
fprintf (stdout,“Enter two ints in this line: ”);
fscanf(stdin,“%d %d”, &first, &second);
fprintf(stdout,“Their sum is: %d.\n”, first + second);
return 0;
}
```

String Manipulation

- C has the weakest character string capability for any general-purpose programming language.
- Strictly speaking, there are no character strings in C, just arrays of single characters that are really small integers.
- If s1 and s2 are such ‘strings’ a program cannot
  - assign one to the other: s1 = s2;
  - compare them for collating sequence: s1 < s2;
  - concatenate them to form a single longer string: s1 + s2;
  - return a string as the result of a function.

String Manipulation

Table String manipulation functions available in `string.h`

Function	Description
`strcpy(s1,s2)`	Copies `s2` into `s1`
`strcat(s1,s2)`	Concatenates `s2` to `s1`. That is, it appends the string contained by `s2` to the end of the string pointed to by `s1`. The terminating null character of `s1` is overwritten. Copying stops once the terminating null character of `s2` is copied.
`strncat(s1,s2,n)`	Appends the string pointed to by `s2` to the end of the string pointed to by `s1` up to n characters long. The terminating null character of `s1` is overwritten. Copying stops once n characters are copied or the terminating null character of `s2` is copied. A terminating null character is always appended to `s1`.
`strlen(s1)`	Returns the length of `s1`. That is, it returns the number of characters in the string without the terminating null character.
`strcmp(s1,s2)`	Returns 0 if `s1` and `s2` are the same Returns less than 0 if `s1<s2` Returns greater than 0 if `s1>s2`
`strchr(s1,ch)`	Returns pointer to first occurrence ch in `s1`
`strstr(s1,s2)`	Returns pointer to first occurrence `s2` in `s1`

Copying a String into Another

- Since C never lets entire arrays to be assigned, the strcpy() function can be used to copy one string to another.
  - strcpy() copies the string pointed to by the second parameter into the space pointed to by the first parameter.
  - The entire string, including the terminating NUL, is copied and there is no check that the space indicated by the first parameter is big enough.
  - The given code shows the use of the strcpy() function.

```
#include <string.h>
int main()
{
      char s1[] =“Hello, world!”;
      char s2[20];
      strcpy(s2, s1);
      puts (s2);
      return 0;
}
```

Comparing Strings

- Another function, strcmp(), takes the start addresses of two strings as parameters and returns the value zero if the strings are equal.
  - If the strings are unequal, it returns a negative or positive value.
  - The returned value is positive if the first string is greater than the second string and negative if the first string is lesser than the second string.
  - In this context, the relative value of strings refers to their relative values as determined by the host computer character set (or collating sequence).
  - It is important to understand that two strings cannot be compared by simply comparing their start addresses although this would be syntactically valid.

Following Program Illustrates the Comparison of Two Strings

```
#include <stdio.h>
#include <string.h>
int main()
{
   char x[50],y[]="a programming example";
   strcpy(x,"A Programming Example");
   if(strcmp(x,"A Programming Example") == 0)
     printf("Equal \n");
   else
     printf("Unequal \n");
   if( strcmp(y,x) == 0)
     printf("Equal \n");
   else
     printf("Unequal \n");
   return 0;
}
```

It produces the following output.

```
Equal
Unequal
```

Contd.

- Since C never lets entire arrays to be assigned, the strcpy() function can be used to copy one string to another.
- Strings can be compared with the help of strcmp() function.
- Arithmetic addition cannot be applied for joining two or more strings; this can be done by using the standard library function, strcat().

Putting Strings Together

- The arithmetic addition cannot be applied for joining of two or more strings in the manner
  - string1 = string2 + string3; or
  - string1 = string2 +”RAJA”;
  - For this, the standard library function, strcat(), which concatenates strings is needed. It does not concatenate two strings together and give a third, new string.
  - In this example, the first call to printf prints “Hello,”, and the second one prints “Hello, world!”, indicating that the contents of str have been appended to the end of s.

```
#include <stdio.h>
#include <string.h>
int main()
{
  char s[30] =“Hello,”;
  char str[] =“world!”;

  printf(“%s\n”, s);
  strcat(s, str);

  printf(“%s\n”, s);
  return 0;
}
```

Multidimensional Arrays

- Arrays with more than one dimension are called multidimensional arrays.
- An array of two dimensions can be declared as follows:
  - **data_type array_name[size1][size2];**
  - Here, data_type is the name of some type of data, such as int. Also, size1 and size2 are the sizes of the array's first and second dimensions, respectively.
- A three-dimensional array, such as a cube, can be declared as follows:
  - **data_type array_name[size1][size2][size3]**

Contd.

- The number of subscripts determines the *dimensionality* of an array. For example, x[i] refers to an element of a one-dimensional array, x. Similarly, y[i][j] refers to an element of a two-dimensional array, y, and so on.
  - int b[3][5]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
- **Unsized Array Initializations**
  - C compiler automatically creates an array big enough to hold all the initializers. This is called an unsized array.
  - The following are examples of declarations with initialization.
    - **char e1[] =“read error\n”;**
    - **char e2[] =“write error\n”;**

Contd.

- Multi-dimensional arrays are kept in computer memory as a linear sequence of variables.
- The elements of a multi-dimensional array are stored contiguously in a block of computer memory.
- The number of subscripts determines the *dimensionality* of an array.
- The separation of initial values into rows in the declaration statement is not necessary.
- If unsized arrays are declared, the C compiler automatically creates an array big enough to hold all the initializers.

```
(a) int a[6][2] = {
                    1,1,
                    2,4,
                    3,9,
                    4,16,
                    5,25,
                    6,36
                  };
(b) int b[3][5] = {{1,2,3,4,5},
                   {6,7,8,9,10},
                   {11,12,13,14,15}
                  };
```

```
int val[3][4] = {8, 16, 9, 52,
                 3, 15, 27, 6,
                 14, 25, 2, 10};
```

```
int x[4][2] = {
              { 1, 2 },
              { 3, 4 },
              { 5, 6 }
              };
```

```
int sgrs[][2] =
{
  1,1,
  2,4,
  3,9,
  4,16,
};
```

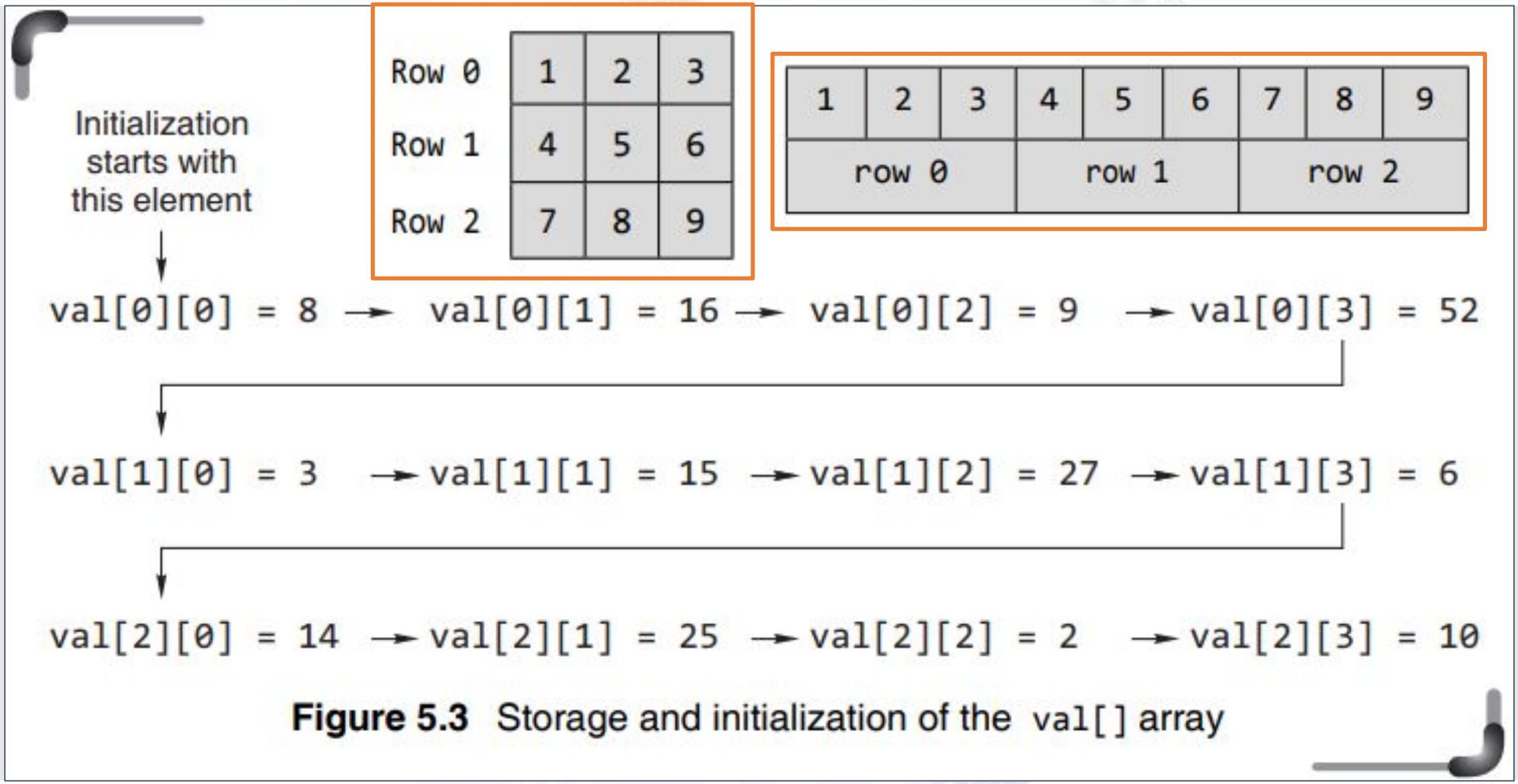


Figure 5.3 Storage and initialization of the `val[]` array

CODE : Transpose of a Matrix

```
#include <stdio.h>
int main()
{
  int row,col;
  int i, j, value;
  int mat[10][10], transp[10][10];
  printf("\n Input the number of rows:");
  scanf("%d", &row);
  printf("Input number of cols:");
  scanf("%d", &col);
  for(i = 0 ; i < row; i++)
  {
    for(j = 0 ; j < col; j++)
    printf("Input Value for : %d: %d:",
          i+1,j+1);
    scanf("%d", &value);
    mat[i][j] = value;
  }
}
printf("\n Entered Matrix is as follows:\n"
for(i = 0; i < row; i++)
{
 for(j = 0; j < col; j++)
 {
  printf("%d", mat[i][j]);
 }
   printf("\n");
 }
 for(i = 0; i< row; i++)
 {
 for(j = 0; j < col; j++)
 {
  transp[i][j]= mat[j][i];
 }
 }
  printf("\n Transpose of the matrix is as\
        follows:\n");
  for(i = 0; i < col; i++)
  {
    for(j = 0; j < row; j++)
   {
    printf("%d", transp[i][j]);
    }
    printf("\n");
   }
     return 0;
   }
```

Arrays of Strings: Two-dimensional Character Array

- A two-dimensional array of strings can be declared as follows:

  <data_type> <string_array_name>[<row_size>] [<columns_size>];

- Consider the following example on declaration of a two-dimensional array of strings.

```
char s[5][30];
```

Initialization

- Two-dimensional string arrays can be initialized as shown
  - char s[5][10] ={“Cow”, “Goat”, “Ram”, “Dog”, “Cat”};
- which is equivalent to
  - s[0] C o w \0
  - S[1] G o a t \0
  - S[2] R a m \0
  - S[3] D o g \0
  - S[4] C a t \0
- Here every row is a string. That is, s[i] is a string. Note that the following declarations are invalid.
  - char s[5][] ={“Cow”,“Goat”,“Ram”,”Dog”,“Cat”};
  - char s[][] ={“Cow”,“Goat”,“Ram”,“Dog”,“Cat”};

Thank You!