Array

- **Character Strings as Arrays of Characters**
- **Array as Function Arguments**
- **String-Handling Functions**
- **Real World Application: Computing A String’s Length**

Character String: Array of Characters

```
char solar1[4];

solar1[0]=‘s’;
solar1[1]=‘u’;
solar1[2]=‘n’;
solar1[3]=‘\0’; /*note: this null character must always be
    present to mark the end of the string; it is one
    character*/
```

Initialize with scanf

```
char solar2[8];
```

```
scanf(“%s”,solar2);/*line 1*/
```

solar2 is identical to &solar2[0]

input

mercury

7 characters+carriage return

Note: use solar2 in scanf; not &solar2

line 1 is equivalent to scanf(“%s”,&solar2[0])

Initialize with double quotes

```
char solar3[5]=“mars”;
```

Note the difference between ‘m’ and “m”

‘m’ represents one character m; one cell

“m” represents a string ‘m’ and ‘\0’; two cells

Printing strings

```
printf( “%s”, solar1 );
```

or

```
printf( “%s”, &solar1[0] );
```

Note: solar1 specifies the address where the string is stored

solar1[0] is a variable. &solar1[0] gives the address.

Array as a function argument

int sum(int a[], int n)

Note: a[], no number of cells

action: sum a[0] through a[n-1]

x = sum (b,m); /*b is a defined array name*/

action: sums b[0] through b[m-1]

equivalent to:

x = sum(&b[0], m);

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_ELEMENTS 100
main ( ) {
        int b[ MAX_ELEMENTS ], x,  m = 0;
        while ( m < MAX_ELEMENTS && scanf ( “%d”, &b[ m ] ) != EOF )
                m++;
        /* reset m so it is the index of the last item in the array b */
        m--;
        printf ( “%d item(s) input\n”, m + 1 );
        if( m >= 0 ) {
                x = sum( b, m );
                printf ( “sum = %d\n”, x );
        }
        return EXIT_SUCCESS;
}
/* A function to sum an array */
int sum( int a[ ], int n ) {
        int partial_sum = 0, I;
        for( I = 0; I <= n; I++)
                partial_sum += a[ I ];
        return partial_sum;
}
```

String handling functions

#include <string.h>

strcat, strncat

Concatenate (joining) 2 strings

strcat(string1, string2):

- 2 character strings as arguments
- joins string1 and string2
- result stored in string1
- returns the address of string1

strncat

strncat(string1, string2, n):

2 character strings and an integer as arguments

joins string1 and n characters in string2

result stored in string1

returns the address of string1

strncat (s1 , s2, n) -- Concatenates at most **n** characters from **s2** to the end of **s1**, returning **s1**.

Where s1, s2 are the character array’s name or pointers to char.

char string1[16] = “I am “;

char string2[11] = “a student.”

strcat (string1, string2);

printf (“The concatenated string is : \t%s”, string1);

the output is

The concatenated string is: I am a student**.**

string1	‘I’	‘ ’	‘a’	‘m’	‘ ’	‘\0’	‘\0’	‘\0’	‘\0’	‘\0’	‘\0’	‘\0’	‘\0’	‘\0’	‘\0’	‘\0’

string2	‘a’	‘ ’	‘s’	‘t’	‘u’	‘d’	‘e’	‘n’	‘t’	‘.’	‘\0’

Initialized arrays.

string1	‘I’	‘ ’	‘a’	‘m’	‘ ’	‘a’	‘ ’	‘s’	‘t’	‘u’	‘d’	‘e’	‘n’	‘t’	‘.’	‘\0’

Using strcat.

If we use strncat,

```
strncat ( string1, string2, 9 );
printf ( “The concatenated string is : \t%s”, string1 );
```

the output is:

```
The concatenated string is:     I am a student
```

string1	‘I’	‘ ’	‘a’	‘m’	‘ ’	‘a’	‘ ’	‘s’	‘t’	‘u’	‘d’	‘e’	‘n’	‘t’	‘\0’	‘\0’

Using strncat.

strcmp,strncmp

Compares two strings

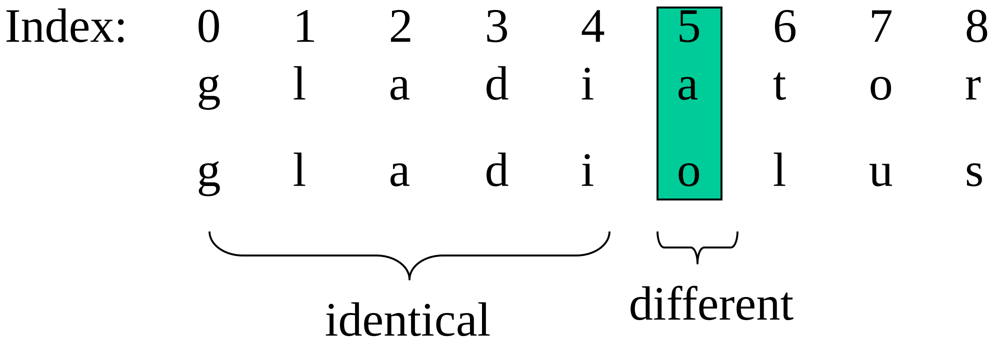
strcmp(string1,string2)

returns

- 0 if two strings are identical
- negative integer if string1<string2
- positive integer if string1 >string 2
- > or < means
  - leftmost position where they differ, p;
  - order of string=order of characters at p according to encoding table
  - if string1 shorter than string2, and each char in string1 is identical to string2, then string1<string2

Example:

gladiator precedes the **gladiolus.**



Index:	0	1	2	3	4	5	6	7	8
	g	l	a	d	i	a	t	o	r
	g	l	a	d	i	o	l	u	s

identical

different

Lexicographic order

‘a’ precedes ‘o’ therefore

order of string1 < order of string2

strncmp

Same as strcmp

strncmp(string1, string2, n)

compares (up to) first n characters

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
main( ) {
        char string1[ ] =  “gladiator”;
        char string2[ ] =  “gladiolus”;
        /* test strcmp function */
        if( strcmp( string1, string2 ) > 0 )
                printf( “using strcmp( string1, string2 ), string1 > string2\n” );
        else if( strcmp( string1, string2 ) == 0 )
                printf( “using strcmp( string1, string2 ), string1 == string2\n” );
        else
                printf( “using strcmp( string1, string2 ), string1 < string2\n” );
        /* test strncmp function */
        if( strncmp( string1, string2, 5 ) > 0 ) /* the first five characters are equal. */
                printf( “using strncmp( string1, string2, 5 ), string1 > string2\n” );
        else if( strncmp( string1, string2, 5 ) == 0 )
                 printf( “using strncmp( string1, string2, 5 ), string1 == string2\n” );
        else
                 printf( “using strncmp( string1, string2, 5 ), string1 < string2\n” );
        return EXIT_SUCCESS;
}
```

The output is

```
using strcmp( string1, string2 ), string1 < string2
using strncmp( string1, string2, 5 ), string1 == string2
```

strcpy, strncpy

Strcpy(string1,string2)

- Copies all or part of string2 to string1
- returns string1

Strncpy(string1,string2,n)

- Copies n characters of string2 to string1
- If string1 is longer than string2, null terminators fill string1

```
#include < stdio.h >
#include < stdlib.h >
#include < string.h >
main( ) {
        char string1[ ] = “My One and Only”;
        char string2[ ] = “South Pacific”;
        strcpy( string1, string2 );
```

```
char string1[ ] = “My One and Only”;
char string2[ ] = “South Pacific”;
```

string1	‘M’	‘y’	‘ ’	‘O’	‘n’	‘e’	‘ ’	‘a’	‘n’	‘d’	‘ ’	‘O’	‘n’	‘l’	‘y’	‘\0’

string2	‘S’	‘o’	‘u’	‘t’	‘h’	‘ ’	‘P’	‘a’	‘c’	‘i’	‘f’	‘i’	‘c’	‘\0’

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

string1	‘S’	‘o’	‘u’	‘t’	‘h’	‘ ’	‘P’	‘a’	‘c’	‘i’	‘f’	‘i’	‘c’	‘\0’	‘y’	‘\0’

string2	‘S’	‘o’	‘u’	‘t’	‘h’	‘ ’	‘P’	‘a’	‘c’	‘i’	‘f’	‘i’	‘c’	‘\0’

```
strcpy( string1, “My One and Only” );
```

string1	‘M’	‘y’	‘ ’	‘O’	‘n’	‘e’	‘ ’	‘a’	‘n’	‘d’	‘ ’	‘O’	‘n’	‘l’	‘y’	‘\0’

```
strncpy( string1, string2, 5 );
```

string1	‘S’	‘o’	‘u’	‘t’	‘h’	‘e’	‘ ’	‘a’	‘n’	‘d’	‘ ’	‘O’	‘n’	‘l’	‘y’	‘\0’
string2	‘S’	‘o’	‘u’	‘t’	‘h’	‘ ’	‘P’	‘a’	‘c’	‘i’	‘f’	‘i’	‘c’	‘\0’		

strlen

strlen(string)

returns length of the string, not counting the null terminator

```
char string[ ] = “Follies”;
char null_string[ ] = “”;
printf( “ %d\n”, strlen( string ) );
printf( “ %d\n”, strlen( null_string ) );
```

the output is

```
7
0
```

Searching a string

strstr(string1, string2)

returns the sub-string of string1 that contains string2

strchr(string,c)

returns the sub-string of string that contains the first character c

strrchr(string,c)

returns the sub-string of string that contains the last character c

returns NULL if search fails

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
main(  ) {
        printf( “%s\n”, strstr( “photon spin”, “on sp” ) );


        printf( “%s\n”, strchr( “photon spin”, ‘n’ );


        printf( “ %s\n”, strrchr( “photon spin”, ‘n’) );
```

output:

```
on spin
n spin
n
```

➢Real World Application: Computing A String’s Length

- ***Problem***

  Write a function to determine a character string’s length.

- ***Sample Input/Output***

Input	Output
“otter”	5
“”	0
“a”	1

- ***Solution***

♦ *C Implementation*

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
 int length( char s[ ] );
main( ) {
        char s1[ ] = “otter”, s2[ ] = “”, s3[ ] = “a”;
        printf( "Input\t\tOutput\n");
        printf( “\”%s\”\t\t%d”, s1, length( s1 ) );
        printf( “\”%s\”\t\t%d”, s2, length( s2 ) );
        printf( “\”%s\”\t\t%d”, s3, length( s3 ) );
        return EXIT_SUCCESS;
}
int length( char s[ ] ) {
        int count;          /* string’s length */
        for( count = 0; s[ count ] != ‘\0’; count++ );
        return count;
}
```

The output is

Input	Output
“otter”	5
“”	0
“a”	1

```
printf( “\”%s\”\t\t%d”, s1, length( s1 ) );
```

The other way:

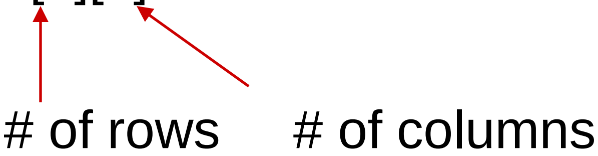
```
printf( “\”%s\”\t\t%d”, s1, strlen( s1 ) );
```

we don’t need the ‘length’ function

Multidimensional Arrays

2-dimensional arrays

```
float a[3][5]
int c[4][4]
```



# of rows   # of columns

3-dimensional arrays

```
float u[10][11][999]
```

Array Definition	***Dimensions***	***Number of Cells***
int tape[100];	one	100
int cars[10] [10];	two	100
char address[100] [10]	two	1,000
float temperature[10] [10] [10]	three	1,000
int count[20] [10] [10]	three	2,000

Initialize an array

- int a[2][3]={{2,22,100},{101,-3,8}};
  - note the double curly brackets
  - first row values followed by second row values
  - array of array; 2-element array of 3-element array

- The array is stored like:
  - 1st row, second row
  - 2,22,100,101,-3,8
  - a[0][0]. a[0][1],a[0][2],a[1][0],a[1][1],a[1][2]
  - the second index changes first and faster

How array elements are stored

- Multi-dimensional array m[constant1][c2][c3]
  - the last index is changing first and faster than C2
  - m[0][0][0], m[0][0][1],m[0][0][2]…,m[0][0][c3-1],
  - m[0][1][0], m[0][1][1],m[0][1][2]…,m[0][1][c3-1],

Multi-dimensional array as arguments

- int A[100][9];
- fun(A);
- void fun(int A[][9]) /*header*/
  - the number of cells must be specified in dimension > 1

Notice:

To declare a parameter for a multidimensional array, we must specify the number of cells in all dimensions beyond the first.

Note: Every array, no matter how many dimensions it has, is implemented as a one-dimensional array.

Note: The size of an array dimension must be given as a constant.

```
void add ( float a[ ] [n], float b[ ] [ n ],           /* Error: n is not a constant */
                                      float c[ ] [n], int n )
```

Matrix multiplication

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \text{x} \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

$$c_{21} = a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31}$$

Z[m][n] = X[m][k] x Y[k][n]

$$Z(i, j) = \sum_{k=0}^{n-1} X(i,k) * Y(k, j)$$

where

$i = 0,\ldots, (m-1)$
$j = 0,\ldots, (n-1)$

```
/* matrix multiplication program */
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE        20
store( int m[ ] [MAXSIZE ], int n );
mult(  int m1[ ] [ MAXSIZE ], int m2[ ] [ MAXSIZE ],
       int m3[ ] [ MAXSIZE ], int n );
print( int m[ ] [ MAXSIZE ], int n );
/*function declaration*/
```

```
main ( ) {
        int n;
        int m1[ MAXSIZE ] [ MAXSIZE ], m2[ MAXSIZE ] [ MAXSIZE ],
                 m3[ MAXSIZE ] [ MAXSIZE ];/*array declaration*/
        /* read data into m1 and m2 and echo */
        printf( “Input matrix size: “ );
        scanf( “%d”, &n );
        printf( “Input first matrix by row\n”);
        store( m1, n );
        printf( “\nMatrix m1: \n” );
        print( m1, n );
        printf( “Input second matrix by row\n”);
        store( m2, n );
        printf( “\nMatrix m2: \n” );
        print( m2, n );
        mult( m1, m2, m3, n );    /* Multiply m1 by m2,  storing product in m3 */
        printf( “\nProduct m3: \n” );
        print( m3, n );                /* print results */
        return EXIT_SUCCESS;
}
```

```
/* Store data in matrix by row */
store( int m[ ] [MAXSIZE ], int n )
{
        int I, j;
        for( I = 0; I < n; I++ )
                for( j = 0; j < n; j++ )
                        scanf( “%d”, &m[ I ] [ j ] );
}
```

```
 mult( int m1[ ] [ MAXSIZE ], int m2[ ] [ MAXSIZE ],
        int m3[ ] [ MAXSIZE ], int n )
{
        int I, j, k;
        for( I = 0; I < n; I++ )
         for( j = 0; j < n; j++ )
         {     m3[ I ] [ j ] = 0;
               for( k = 0; k < n; k++ )
                       m3[ I ] [ j ] += m1[ I ] [ k ] * m2[ k ] [ j ];
         }
}
print( int m[ ] [ MAXSIZE ], int n ) {
        int I, j;
        for( I = 0; I < n; I++ ) {
         for( j = 0; j < n; j++ )
               printf( “%d “, m[ I ] [ j ] );
         printf( “\n” );
        }
}
```

➢ Real World Application:

Solving A Linear System of Equations

- ***Problem***

$$\begin{aligned}
&2x + 2y -2z = -6\\
&4x + 7y + 3z = 3\\
&6x + 12y + z = -9
\end{aligned}$$

Gaussian elimination

$$\begin{aligned} \mathbf{m_{00}x + m_{01}y + m_{02}z = m_{03}} \\ \mathbf{m_{11}y + m_{12}z = m_{13}} \\ \mathbf{m_{22}z = m_{23}} \end{aligned}$$

$$\begin{array}{lr} 2x + 2y - 2z = -6 & (1) \\ 4x + 7y +3z = 3 & (2) \\ 6x + 12y + z = -9 & (3) \end{array}$$

equation (2) + equation (1) * (- 4/2), we get

$$\begin{array}{rr} -4x - 4y + 4z = 12 & \\ \underline{4x +7y + 3z = 3} & \\ 3y + 7z = 15 & (2') \end{array}$$

Equation (3) + equation (1) * (- 6/2), we get

$$\begin{array}{ll} 6x + 12y + z = -9 & \\ \underline{-6x - 6y + 6z} = 18 & \\ 6y + 7z = 9 & (3') \end{array}$$

Our system of equations becomes

$$\begin{array}{rl} 2x + 2y - 2z = -6 & (1) \\ 3y + 7z = 15 & (2') \\ 6y + 7z = 9 & (3') \end{array}$$

Equation (3') + equation (2') * (- 6/3), we get

$$\begin{array}{r} 2x + 2y - 2z = -6 \\ 3y + 7z = 15 \\ - 7z = -21 \end{array}$$

The solution is:

$z = 3$, $y = -2$, $x = 2$

```
        m[0] [0], m[0] [1], …, m[0] [n-1]
        m[1] [0], m[1] [1], …, m[1] [n-1]
        …
        m[n-1] [0], m[n-1] [1], …, m[n-1] [n-1]
        the constants are stored as
        m[0] [n], m[1] [n], …, m[n-1] [n]
*/
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 15
void linear_solve( float m[] [MAXSIZE + 1], float x[ ], int n );
main( ) {
        float workspace[ MAXSIZE ] [ MAXSIZE + 1], x[ MAXSIZE ];
        int I, j, n;
        printf( “Number of equationis? “ );
        scanf( “%d”, &n );
        printf( “\n\nEnter coefficients by rows--\n” );
```

```
    for( I = 0; I < n; I++ )
        for( j = 0; j < n; j++ )
            scanf( “%f”, &workspace[ I ] [ j ] );
    printf( “\n\nEnter constants--\n” );
    for( I = 0; I < n; I++ )
        scanf( “%f”, &workspace[ I ] [ n ] );
    /* Solving using Gaussian elimination */
    linear_solve( workspace, x, n );
    printf( “\n\nSolution--\n” );
    for( I = 0; I < n; I++ )
        printf( “\tx[ %d ] = %f\n”, I, x[ I ] );
    return EXIT_SUCCESS;
}
```

```
/* Solving using Gaussian elimination */
void linear_solve( float m[] [MAXSIZE + 1], float x[ ], int n )
{
        int I, j, k, pivot;
        float factor, temp;
        for( I = 0; I < n; I++ )
        {
          if( m[ I ] [ I ] == 0.0 ) {
                pivot = 0;
        for( j = I + 1; j < n; j++ ) /* find next nonzero entry in col I */

        if( m[ j ] [ I ] != 0.0 ) {
                pivot = j;
                break;
        }
        /* if no nonzero entry in column I, system is singular */
        if( pivot == 0 ) {
                printf( “System is singular\n” );
                exit (EXIT_FAILURE );
        }
        /* swap so m [ I ] [ I ] != 0 */
        for( j = 0;  j < n + 1; j++ ) {
                temp = m[ I ] [ j ];
                m[ I ] [ j ] = m[ pivot ] [ j ];
                m[ pivot ] [ j ] = temp;
        }
}
```

```
        /* make column I, row j >= I + 1, zero */
        for( j = I + 1; j < n; j++ )  {
            factor = -m[ j ] [ I ] / m[ I ] [ I ];
            for( k = I; k < n + 1; k++ )
                m[ j ] [ k ] += factor * m[ I ] [ k ];
        }
    }
    /* solve for unknowns */
    x[ n - 1 ] = m[ n - 1 ] [ n ] / m[ n - 1 ] [ n - 1 ];
    for( j =  n - 2; j >= 0; j-- ) {
        x[ j ] = m[ j ] [ n ];
        for( k = j + 1; k < n; k++ )
            x[ j ] -= m[ j ] [ k ] * x[ k ];
        x[ j ] /= m[ j ] [ j ];
    }
}
```

- ***Discussion***