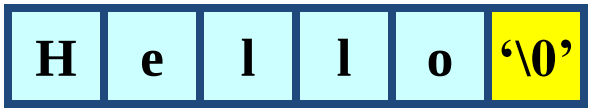
Arrays- Part 2

Character String

Introduction

- A string is an array of characters.
  - Individual characters are stored in memory in ASCII code.
  - A string is represented as a sequence of characters terminated by the null (‘\0’) character.
  - Because C stores a string as an array, the name of the string is a pointer to the beginning of the string.

“Hello” →



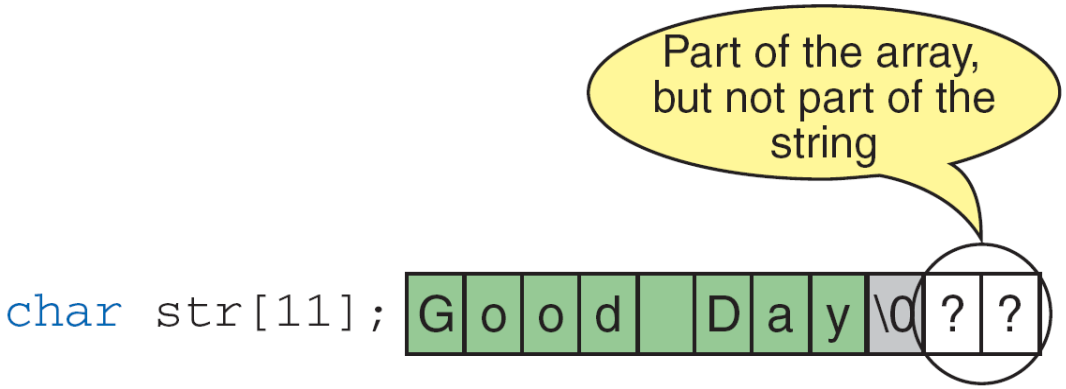
  - C implements strings logically, not physically. The physical structure of a string is the array in which C stores the string.

Declaring String Variables

- A string is declared like any other array:

  char string-name [size];

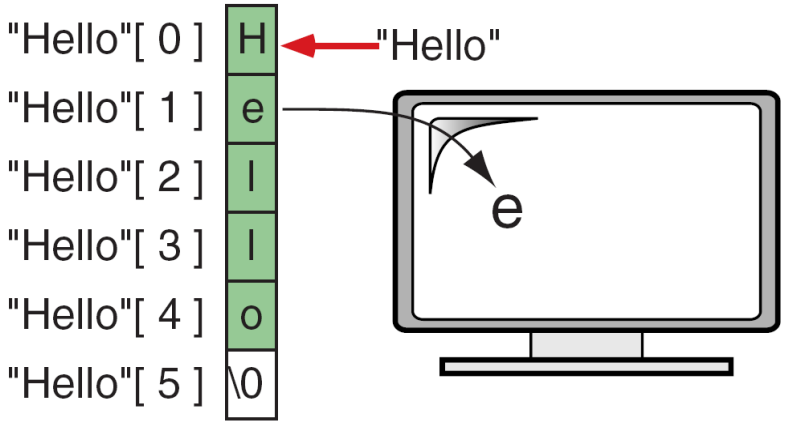
  – size determines the number of characters in string_name.
- We must provide enough room for the maximum number of characters in a string. However, if we don't fill the string, C puts the null character in the middle of the string and ignores the remaining spaces in the array.



String Literals

- String literals, aka “string constants”, are sequences of characters enclosed in double quotes.
- C automatically creates a null-delimited string array when it encounters characters in double quotes.
- Referencing a string literal:

```
#include <stdio.h>
int main (void)
{
  printf("%c\n", "Hello"[1];
  return 0;
} // main
```



Declaring Strings

- We can declare strings as a character array with a length of maximum characters needed + 1.
  ```
  char name[21];
  ```
- We can also declare a string as a pointer. However, in this case, unlike the first method, C doesn't allocate memory for the string:
  ```
  char* pName;
  ```
- We must allocate memory for strings before we use them!

Initializing Strings

- We can initialize by assigning a string literal:
```
char myString[13] = "Hello World!";
char myString[] = "Hello World!";
```
- Assigning a string literal to a pointer:
```
char* pMyString = "Hello World!";
```
- We can also initialize a string as an array of characters (however, this isn't used too often, as it is cumbersome).

Strings & The Assignment Operator

- The name of a string is a pointer constant to the first character in the character array. As such, we need to take great care when assigning values to a string.
- The following results in an error:

```
char str1[6] = "Hello";
char str2[6];
str2 = str1; //Results in Error
```

Reading Strings

- Two different cases will be considered:
  - Reading words
  - Reading an entire line

Reading “words”

- scanf can be used with the “%s” format specification.

```
char   name[30];
:
:
scanf (“%s”, name);
```

  - The ampersand (&) is not required before the variable name with “%s”.
  - The problem here is that the string is taken to be upto the first white space (blank, tab, carriage return, etc.)
    - If we type “Rupak Biswas”
    - name will be assigned the string “Rupak”

Reading a “line of text”

- In many applications, we need to read in an entire line of text (including blank spaces).
- We can use the getchar() (or scanf (“%c”, …) for the purpose.

```
char   line[81], ch;
int  c=0;
:
:
do {
      ch = getchar();
      line[c] = ch;
      c++;
  }
while  (ch != '\n');

c = c – 1;
line[c] = '\0';
```

Read characters until CR ('\n') is encountered

Make it a valid string

Reading a line :: Alternate Approach

```
char   line[81];
:
scanf  (“%[  ABCDEFGHIJKLMNOPQRSTUVWXYZ]”, line);
```

Reads a string containing uppercase characters and blank spaces

```
char   line[81];
:
scanf  (“%[^\n]”, line);
```

Reads a string containing any characters

The `gets()` Function

- The `gets()` function (from `stdio`) takes a string from standard input and assigns it to a character array. It replaces the `\n` with `\0`.
- To use `gets()`:
  ```
  gets(myString);
  ```
- The `gets()` function includes no way to check the length of the input string!

Writing Strings to the Screen

- We can use printf with the “%s” format specification.

```
char name[50];
:
:
printf (“\n %s”, name);
```

Processing Character Strings

- There exists a set of C library functions for character string manipulation.
  - strcpy :: string copy
  - strlen :: string length
  - strcmp :: string comparison
  - strtcat :: string concatenation
- It is required to include the following

```
#include <string.h>
```

Copying strings

- We cannot simply assign one string to another due to the that strings are character arrays!
- strcpy () Works very much like a string assignment operator.

  strcpy (destinationString, sourceString);

- Examples:

  strcpy (city, “Calcutta”);

  strcpy (city, mycity);

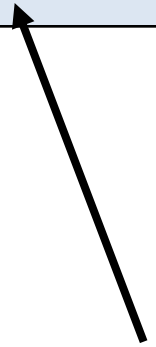
strlen()

- Counts and returns the number of characters in a string.

```
len  =  strlen (string);   /* Returns an integer */
```

  - The null character (‘\0’) at the end is not counted.
  - Counting ends at the first null character.

```
char  city[15];
int  n;
:
:
strcpy (city, “Calcutta”);
n = strlen (city);
```



n is assigned 8

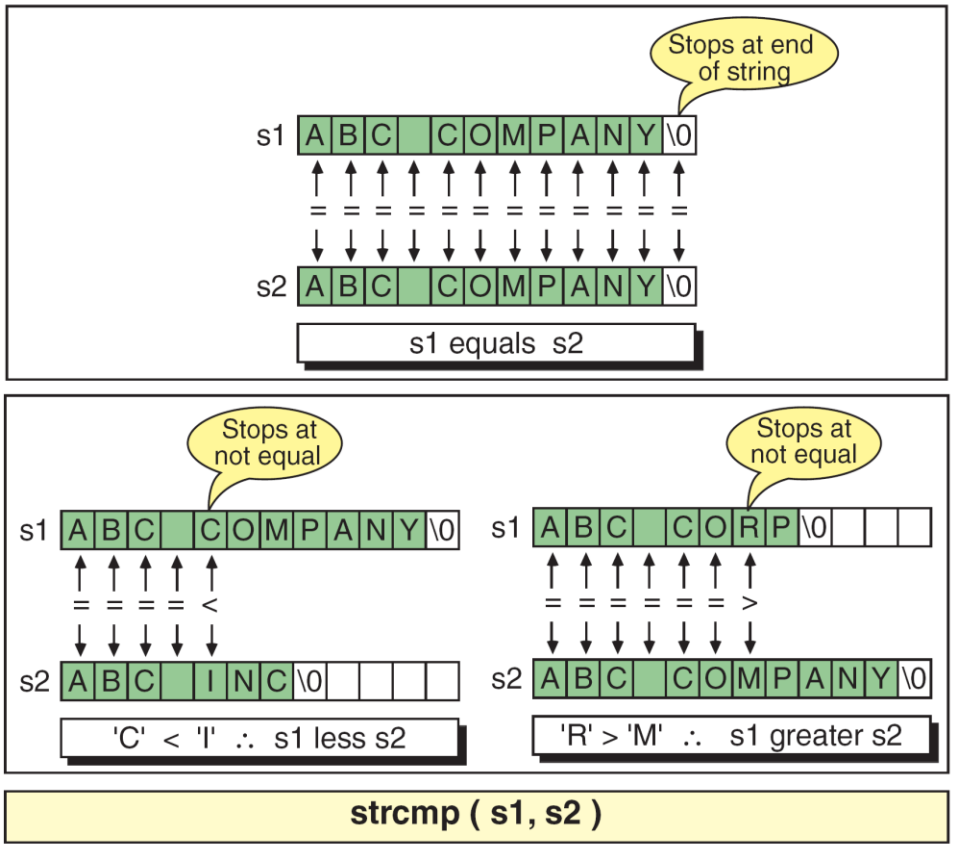
Comparing Strings: strcmp()

- Compares two character strings.

  int strcmp (str1, str2);

- If the two strings are equal, **`strcmp()`** returns **`0`**.
- If **`str1`** is greater than **`str2`**, **`strcmp()`** returns a positive number.
- If **`str1`** is less than **`str2`**, **`strcmp()`** returns a negative number.

Comparing Strings



Combining Strings: strcat()

- Joins or concatenates two strings together.

  strcat (string1, string2);

  – string2 is appended to the end of string1.

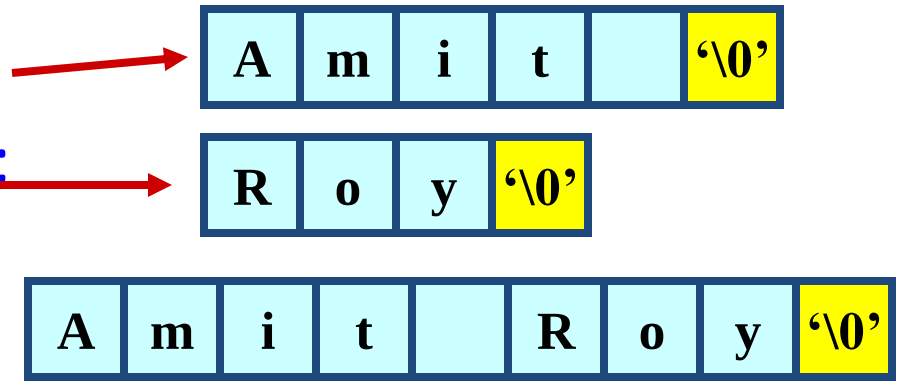
  – The null character at the end of string1 is removed, and string2 is joined at that point.

- Example:

  strcpy (name1, “Amit “);

  strcpy (name2, “Roy“);

  strcat (name1, name2);



```
/* Read a line of text and count the number of uppercase letters */
#include  <stdio.h>
#include  <string.h>
main()
{
   char  line[81];
   int  i, n, count=0;
   printf("Input the line \n");
   scanf ("%[^\n]", line);
   n = strlen (line);
   for  (i=0; i<n; i++)
     {
        if  (isupper (line[i]))
             count++;
     }
   printf ("\n The number of uppercase letters in the string %s is %d",
                          line, count);
}
```

Include header for string processing

Character Array for String

Reading a line of text

Computing string length

Checking whether a character is Uppercase

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:

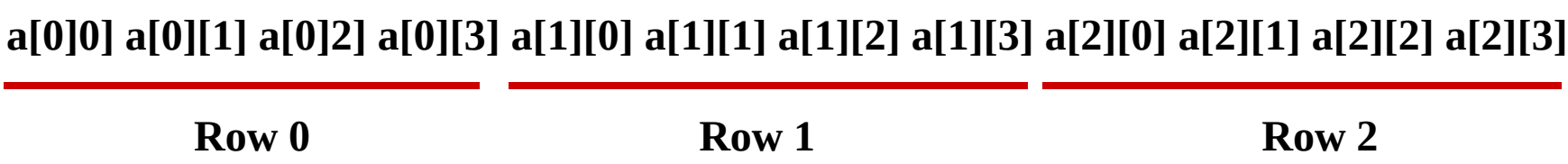
```
x[m][n] = 0;
c[i][k] += a[i][j] * b[j][k];
a = sqrt (a[j*3][k]);
```

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**



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 in matrix form.

Example: Matrix Addition

```
int 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 a Function

- An array name can be used as parameter of a function, which is effectively the address of the first element.
- 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.

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>
int 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];

Number of columns

Example: Transpose of a matrix

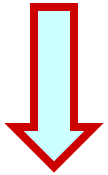
```
void  transpose (int x[][100], int n)
{
   int  p, q;

   for  (p=0; p<n; p++)
      for  (q=0; q<n; q++)
        {
            t = x[p][q];
            x[p][q] = x[q][p];
            x[q][p] = t;
        }
}
```

10 20 30

40 50 60

70 80 90

a[100][100]



transpose(a,3)

10 20 30

40 50 60

70 80 90

The Correct Version

```
void  transpose (int x[][100], n)
{
   int  p, q;

   for  (p=0; p<n; p++)
      for  (q=p; q<n; q++)
         {
            t = x[p][q];
            x[p][q] = x[q][p];
            x[q][p] = t;
         }
}
```