Character String
What we should learn about strings

- Representation in C
- String Literals
- String Variables
- String Input/Output
  - printf, scanf, gets, fgets, puts, fputs
- String Functions
  - strlen, strcpy, strncpy, strcmp, strncmp, strcat, strncat, strchr, strrchr, strstr, strspn, strcspn, strtok
- Reading from/Printing to Strings
  - sprintf, sscanf
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.

“Hello” ➔

H e l l o \0
String Literals

- String literal values are represented by sequences of characters between double quotes ("")

- Examples
  - "" represents empty string
  - "hello"

- "a" versus ‘a’
  - ‘a’ is a single character value (stored in 1 byte) as the ASCII value for the letter, a.
  - "a" is an array with two characters, the first is a, the second is the character value \0.
Referring to String Literals

- String literal is an array, can refer to a single character from the literal as a character

- Example:
  ```c
  printf("%c", "hello"[1]);
  
  outputs the character ‘e’
  ```

- During compilation, C creates space for each string literal (number of characters in the literal + 1)
Duplicate String Literals

• Each string literal in a C program is stored at a different location.
  – Even if the string literals contain the same string, they are not equal (in the == sense)

• Example:

```c
char string1[6] = "hello";
char string2[6] = "hello";

– but string1 does not equal string2 (they are stored in different memory locations).
```
Declaring String Variables

• A string is declared like any other array:

```c
    char string-name[size];
```

  – `size` determines the number of characters in `string_name`.

• When a character string is assigned to a character array, it automatically appends the null character ('\0') at the end of the string.

  – `size` should be equal to the number of characters in the string plus one.
Examples

char name[30];
char city[15];
char dob[11];

- A string may be initialized at the time of declaration.

```
char city[15] = "Calcutta";
char city[15] = {'C', 'a', 'l', 'c', 'u',
                 't', 't', 'a', '\0'};
char dob[] = "12-10-1975";
```
Changing String Variables

• Cannot change string variables connected to string constants, but can change pointer variables that are not tied to space.

• Example:

```c
char *str1 = "hello";    /* str1 unchangeable */
char *str2 = "goodbye";  /* str2 unchangeable */

char *str3;    /* Not tied to space */
str3 = str1;   /* str3 points to same space as str1 */
str3 = str2;
```
Changing String Variables (cont)

- Can change parts of a string variable:

```c
char str1[6] = "hello";
str1[0] = 'y';    /* str1 is now "yello" */
str1[4] = '\0';  /* str1 is now "yell" */
```

- Have to stay within limits of the array.
  - Responsibility of programmer.
Reading Strings from the Keyboard

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

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

```c
char name[30];

scanf ("%s", name);
```

- The ampersand (&) is not required before the variable name with ”%s”.
  - Because `name` represents an address.

- The problem here is that the string is taken to be up to 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()` function for the purpose.
```c
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

```c
char line[81];
:
:
scanf ("%[ ABCDEFGHIJKLMNOPQRSTUVWXYZ]", line);
```

⇒ Reads a string containing uppercase characters and blank spaces

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

⇒ Reads a string containing any characters
More on String Input

- **Edit set input % [ListofChars]**
  - ListofChars specifies set of characters (called scan set)
  - Characters read as long as character falls in scan set
  - Stops when first non scan set character encountered
  - Any character may be specified except ]
  - Putting ^ at the start to negate the set (any character BUT list is allowed)

- **Examples:**
  
  ```c
  scanf ("%[+-0123456789]", Number);
  scanf ("%[^\n]", Line);    /* read until newline char */
  ```
Writing Strings to the Screen

- We can use `printf` with the "%s" format specification.

```c
char name[50];

: 

printf ("\n %s", name);
```
#include <stdio.h>

void main( )
{
    char LastName[11];
    char FirstName[11];

    printf("Enter your name (last, first): ");
    scanf("%10s%[^,],%10s", LastName, FirstName);

    printf("Nice to meet you %s %s\n", FirstName, LastName);
}
String Functions
Processing Character Strings

• There exists a set of C library functions for character string manipulation.
  – `strcpy` :: string copy
  – `strlen` :: string length
  – `strcmp` :: string comparison
  – `strcat` :: string concatenation

• It is required to add the line
  ```
  #include <string.h>
  ```
strcpy()

• Works like a string assignment operator.
  
  char *strcpy (char *str1, char *str2);
  – Assigns the contents of str2 to str1.
  – Returns address of the destination string.

• Examples:
  
  strcpy (city, "Calcutta");
  strcpy (city, mycity);

• Warning:
  – Assignment operator do not work for strings.
  
  city = "Calcutta "; ➔ INVALID
**strlen()**

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

  ```c
  int strlen (char *str);
  ```

- **Example:**

  ```c
  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
**strcmp()**

- **Compares two character strings.**
  ```c
  int strcmp (char *str1, char *str2);
  ```
  - Compares the two strings and returns 0 if they are identical; non-zero otherwise.

- **Examples:**
  ```c
  if (strcmp(city, "Delhi") == 0)
  {
    ...... 
  }
  ```

  ```c
  if (strcmp(city1, city2) != 0)
  {
    ...... 
  }
  ```
• Actually, the function returns the difference in ASCII values of the first letter of mismatch.
  – Less than 0
    • If the ASCII value of the character they differ at is smaller for str1, or str2 is longer than str1
  – Greater than 0
    • If the ASCII value of the character they differ at is greater for str1, or str1 is longer than str2
  – Equal to 0
    • If the two strings are identical
strcmp examples:

- strcmp("hello", "hello") -- returns 0
- strcmp("yello", "hello") -- returns value > 0
- strcmp("Hello", "hello") -- returns value < 0
- strcmp("hello", "hello there") -- returns value < 0
- strcmp("some diff", "some dift") -- returns value < 0

- Expression for determining if two strings s1, s2 hold the same string value:
  
  !strcmp(s1, s2)
String Comparison (strncmp)

Sometimes we only want to compare first n chars:

```c
int strncmp(char *s1, char *s2, int n)
```

Works the same as strcmp except that it stops at the nth character

*looks at less than n characters if either string is shorter than n*

```
strcmp("some diff", "some DIFF")  -- returns value > 0
strncmp("some diff", "some DIFF", 4)  -- returns 0
```
String Comparison (ignoring case)

```c
int strcasecmp(char *str1, char *str2)
```

- similar to strcmp except that upper and lower case characters (e.g., ‘a’ and ‘A’) are considered to be equal

```c
int strncasecmp(char *str1, char *str2, int n)
```

- version of strncmp that ignores case
strcat()

- Joins or concatenates two strings together.

  char *strcat (char *str1, char *str2);
  - str2 is appended to the end of str1.
  - The null character at the end of str1 is removed, and str2 is joined at that point.

- Example:

  ```
  strcpy(name1, "Amit ");
  strcpy(name2, "Roy");
  strcat(name1, name2);
  ```

  ![Diagram showing concatenation of strings]

  ```
  Amit
  Roy
  Amit Roy
  ```
Example:: count uppercase

/* 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;
    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);
}
Example:: compare two strings

```c
#include <stdio.h>

int my_strcmp (char s1[],char s2[])
{
    int i=0;
    while(s1[i]!="\0" && s2[i]!="\0"){
        if (s1[i]!=s2[i]) return(s1[i]-s2[i]);
        else i++;
    }
    return(s1[i]-s2[i]);
}
```

Parameters passed as character array
```c
main()
{
    char string1[100], string2[100];

    printf("Give two strings \n");
    scanf("%s %s", string1, string2);

    printf("Comparison result: %d \n",
            my_strcmp(string1, string2));
}
```

**Give two strings**
IITKGP IITMUMBAI
Comparison result: -2

**Give two strings**
KOLKATA KOLKATA
Comparison result: 0
Searching for a Character/String

char *strchr (char *str, int ch)

- returns a pointer to the first occurrence of ch in str
- returns NULL if ch does not occur in str
- can subtract original pointer from result pointer to determine which character in array

char *strstr (char *str, char *searchstr)

- similar to strchr, but looks for the first occurrence of the string searchstr in str

char *strrchr (char *str, int ch)

- similar to strchr except that the search starts from the end of string str and works backward
Printing to a String

• The `sprintf` function allows us to print to a string argument using `printf` formatting rules.
• First argument of `sprintf` is string to print to, remaining arguments are as in `printf`.

Example:

```c
char buffer[100];
sprintf (buffer, "%s, %s", LastName, FirstName);
if (strlen(buffer) > 15)
    printf("Long name %s %s\n", FirstName, LastName);
```
Reading from a String

- The `sscanf` function allows us to read from a string argument using `scanf` rules.
- First argument of `sscanf` is string to read from, remaining arguments are as in `scanf`.

Example:

```c
char buffer[100] = "A10 50.0";
sscanf (buffer, "%c%d%f", &ch, &inum, &fnum);
/* puts 'A' in ch, 10 in inum and 50.0 in fnum */
```
Example: Duplicate Removal

Write a C function that takes a string as an argument and modifies the string so as to remove all consecutive duplicate characters, e.g., *mississippi* -> *misisipi*

```c
void remove_duplicates (char word[]) {
    int k, j;
    char prev = '\0';
    for (k = j = 0; word[k] != '\0'; k++) {
        if (prev != word[k]) word[j++] = word[k];
        prev = word[k];
    }
    word[j] = '\0';
}
```
2-D Arrays in C

Dept. of Computer Science & Engg.
Indian Institute of Technology
Kharagpur
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.

<table>
<thead>
<tr>
<th></th>
<th>Subject 1</th>
<th>Subject 2</th>
<th>Subject 3</th>
<th>Subject 4</th>
<th>Subject 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student 1</td>
<td>75</td>
<td>82</td>
<td>90</td>
<td>65</td>
<td>76</td>
</tr>
<tr>
<td>Student 2</td>
<td>68</td>
<td>75</td>
<td>80</td>
<td>70</td>
<td>72</td>
</tr>
<tr>
<td>Student 3</td>
<td>88</td>
<td>74</td>
<td>85</td>
<td>76</td>
<td>80</td>
</tr>
<tr>
<td>Student 4</td>
<td>50</td>
<td>65</td>
<td>68</td>
<td>40</td>
<td>70</td>
</tr>
</tbody>
</table>
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 that evaluate to integer values.

• Examples:

\[ x[m][n] = 0; \]
\[ c[i][k] += a[i][j] \times b[j][k]; \]
\[ val = \sqrt{a[j*3][k]}; \]
How is a 2-D array is stored in memory?

- Starting from a given memory location, the elements are stored \textit{row-wise} in consecutive memory locations.
  \begin{itemize}
  \item \textbf{x}: starting address of the array in memory
  \item \textbf{c}: number of columns
  \item \textbf{k}: number of bytes allocated per array element
  \end{itemize}

Element \(a[i][j]:: \text{allocated memory location at address } x + (i \times c + j) \times k\)

\begin{align*}
\text{Row 0} & \quad \text{Row 1} & \quad \text{Row 2} \\
\begin{array}{cccc}
  a[0][0] & a[0][1] & a[0][2] & a[0][3] \\
  a[1][0] & a[1][1] & a[1][2] & a[1][3] \\
\end{array}
\end{align*}
How to read the elements of a 2-D array of size \( nrow \times ncol \)?

- By reading them one element at a time
  
  ```c
  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.

```c
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.

```c
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.

```c
for (i=0; i<nrow; i++)
{
    printf ("\n");
    for (j=0; j<ncol; j++)
        printf ("%f ", a[i][j]);
}
```

- The elements are printed nicely in matrix form.

- How to print two matrices side by side?
• Printing two matrices $A$ and $B$ of sizes $m \times n$ each side by side.

```c
for (i=0; i<m; i++)
{
    printf ("\n");
    for (j=0; j<n; j++)
        printf ("%f ", A[i][j]);
    printf ("\n");
    for (j=0; j<n; j++)
        printf ("%f ", B[i][j]);
}
```
Example: Matrix Addition

```c
#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 ", c[p][q]);
    }
}
```
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

```c
#include <stdio.h>

main()
{
    int a[15][25], b[15][25];
    :
    :
    add (a, b, 15, 25);
    :
}
```

We can also write
```
void add (x, y, rows, cols)
int x[][25], y[][25];
int rows, cols;
{
    :
}
```
```
int x[15][25], y[15][25];
```
Example: Transpose of a matrix

```c
#include <stdio.h>

void transpose (x, n)
int x[][3], n;
{
    int p, q, t;
    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;
        }
}

main()
{
    int a[3][3], p, q;
    for (p=0; p<3; p++)
        for (q=0; q<3; q++)
            scanf ("%d", &a[p][q]);
    transpose (a, 3);
    for (p=0; p<3; p++)
        {
            printf ("\n");
            for (q=0; q<3; q++)
                printf ("%d ", a[p][q]);
        }
}
```
Is the function correct?

10 20 30
40 50 60
70 80 90

10 20 30
40 50 60
70 80 90
The Correct Version

```c
void transpose (x, n)
int x[][3], n;
{
    int p, q, t;

    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;
            }
}
```

```
10 20 30
40 50 60
70 80 90
10 40 70
20 50 80
30 60 90
```
Some Exercise Problems to Try Out

1. A shop stores $n$ different types of items. Given the number of items of each type sold during a given month, and the corresponding unit prices, compute the total monthly sales.

2. Multiple two matrices of orders $mxn$ and $nxp$ respectively.