

Character String

1

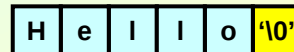
What we should learn about strings

- 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" →



3

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:

```
printf("%c", "hello"[1]);
```


outputs the character 'e'
- During compilation, C creates space for each string literal (# of characters in the literal + 1)

Duplicate String Literals

- Each string literal in a C program is stored at a different location
- So even if the string literals contain the same string, they are not equal (in the == sense)
- Example:
 - `char string1[6] = "hello";`
 - `char string2[6] = "hello";`
 - but `string1` does not equal `string2` (they are stored at different locations)

Declaring String Variables

- A string is declared like any other array:

```
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.

7

Examples

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

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

Equivalent

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

8

Changing String Variables

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

- **Example:**

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

```
char *str3;              /* Not tied to space */
str3 = str1;             /* str3 points to same space s1 connected to */
str3 = str2;
```

Changing String Variables (cont)

- Can change parts of a string variable

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

- Important to retain delimiter (replacing str1[5] in the original string with something other than '\0' makes a string that does not end)
- Have to stay within limits of array

Reading Strings from the Keyboard

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

11

Reading “words”

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

```
char    name[30];
```

```
⋮
```

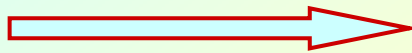
```
scanf ("%s", name);
```

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

12

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.



13

```
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

14

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

15

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
 - Note, does not ignored leading white space
 - Any character may be specified except]
 - Putting ^ at the start to negate the set (any character BUT list is allowed)
- Examples:


```
scanf ("%[-+0123456789]", Number);
scanf ("%[^\\n]", Line);      /* read until newline char */
```

Writing Strings to the Screen

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

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

17

Input / Output Example

```
#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

19

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>`

20

strcpy()

- Works very much 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
```

21

strlen()

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

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

- Example:

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

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

22

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

n is assigned 8

23

strcmp()

- Compares two character strings.

```
int strcmp(char *str1, char *str2);
```

- Compares the two strings and returns 0 if they are identical; non-zero otherwise.

- Examples:

```
if (strcmp(city, "Delhi") == 0)  
{ ..... }
```

```
if (strcmp(city1, city2) != 0)  
{ ..... }
```

24

- **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

25

String Comparison

strcmp examples:

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

expression for determining if two strings s1, s2 hold the same string value:

`!strcmp(s1, s2)`

String Comparison

Sometimes we only want to compare first n chars:

```
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)

```
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

```
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);
```

A	m	i	t		'\0'
---	---	---	---	--	------

R	o	y	'\0'
---	---	---	------

A	m	i	t		R	o	y	'\0'
---	---	---	---	--	---	---	---	------

29

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 ("%s", 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);
}
```

30

Example:: compare two strings

```
#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

Compare character pairs till the end of a string

Return immediately if they are not equal.

31

```
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

32

Searching for a Character/String

char *strchr(char *str, int ch)

- returns a pointer (a char *) 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:

```
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:

```
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.

```
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';
}
```