CHARACTER STRINGS

DR. PAWAN KUMAR SINGH

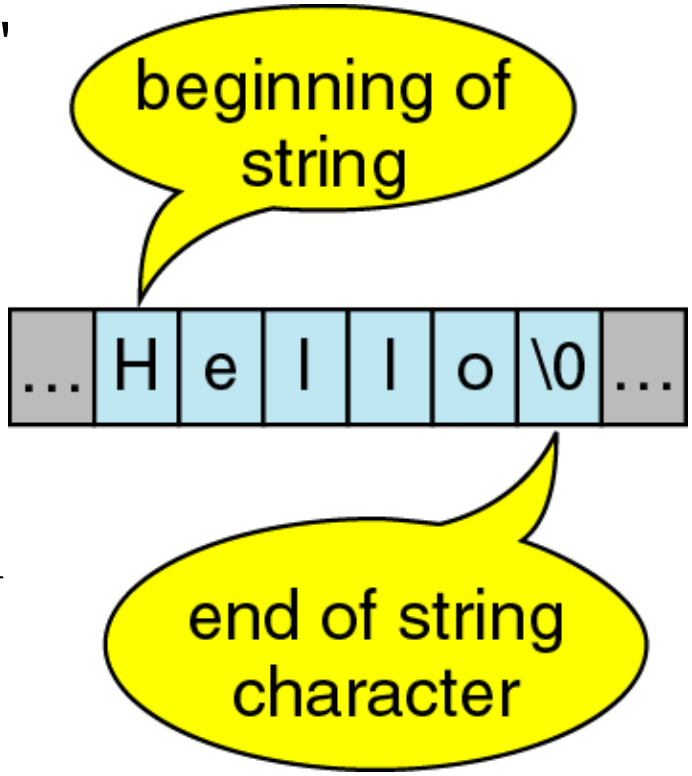
DEPARTMENT OF INFORMATION TECHNOLOGY

JADAVPUR UNIVERSITY

KOLKATA

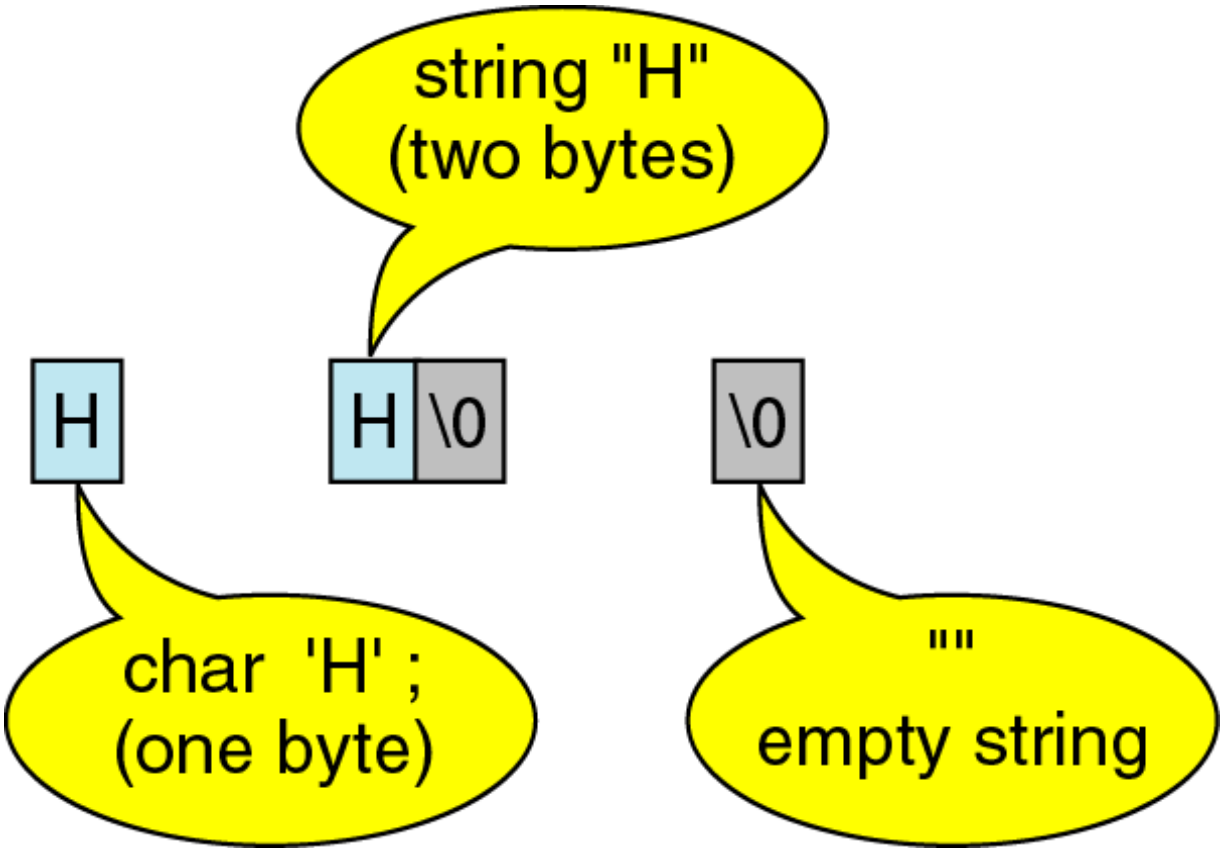
STRING FUNDAMENTALS

- A **string literal** is any sequence of characters enclosed in double quotes
  - Example: `"Good Morning!"`
  - Also called:
    - **string constant**
    - **string value**
    - **string**
  - A string is stored as an array of characters <u>terminated</u> by an <u>end-of-string symbolic constant</u> named NULL (`'\0'`)

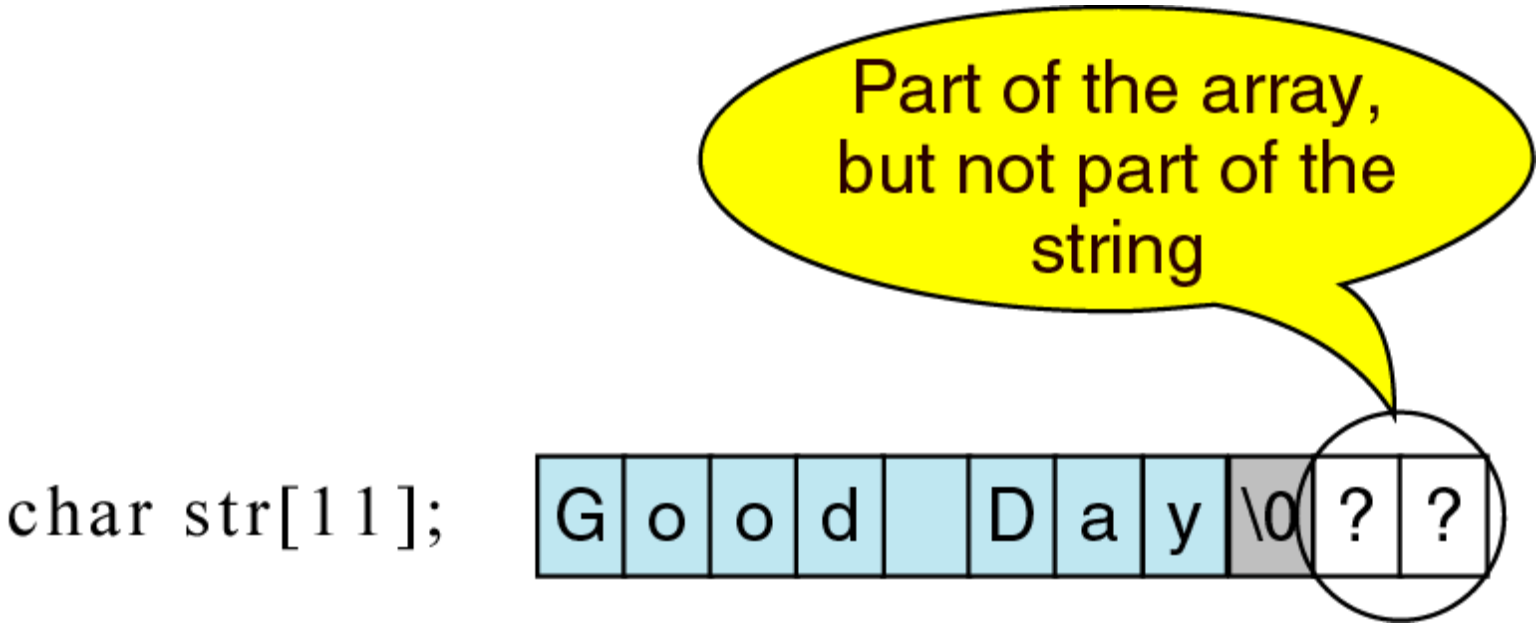


Character Strings

- A **string** containing a single character takes up 2 bytes of storage.



Character Strings



String Fundamentals

- Character literal – a single character in single quotes ‘ ’
  - Example: ‘c’
- String literal – sequence of characters enclosed in double quotes “ ”
  - Example: “This is a string”
- A variable-length array of characters that is delimited by the null character (‘\0’).

Declaration of String

- Example:

```
char symbol; // declare a character
char str[80];            // declare a string
```

- Example

```
#define NUM_STUDENTS 30   }      //defined constants
#define NAME_LEN 25       }

char names [NUM_STUDENTS] [NAME_LEN];
```

Initialization of a String

- To assign value into the string or character
- Example:

```
char message[8];            //declare

message[] = "Shah";                 //assign value
```

OR

```
message[8] = "Shah";
```

OR

```
message[8] ={'S','h','a','h','\0'};
```

Storing string in memory

- String is stored as an array of characters
- Each individual character can be input, manipulated or output
- The end-of-string null character (\0) is used to detect end of the string
- Example of an array with a string “Salam to you”:

S	a	l	a	m		t	o		y	o	u	\0

String Input and Output

- To get input string:
  - gets ( )
  - scanf ( )
  - getchar ( )
  - getch ( )
- To produce string output
  - puts ( )
  - printf ( )
  - putchar ( )
- Note: program must include <stdio.h> file

- **gets**
  - a function that will get a string of characters

```
#include <stdio.h>

int main()
{
        char s[80];
        printf("Please type in some words : ");
        gets(s);
        printf("You typed : %s\n", s);
        return 0;
}
```

- **scanf**
  - to input individual words from a line of text
  - blank spaces act as delimiters

```
#include <stdio.h>

int main()
{
        char word1[80], word2[80];
        printf("Please type TWO words : ");
        scanf("%s %s", &word1, &word2);
        printf("You typed : %s and %s\n", word1, word2);
        return 0;
}
```

- **getchar**
  - to input a single character
  - requires you to hit enter.

```
#include <stdio.h>

int main()
{
        char c;
        printf("Please type ONE character : ");
        c=getchar();
        printf("You typed : %c\n", c);
        return 0;
}
```

- **getch**
  - to input a single character
  - reads a key hit without waiting for you to press enter.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        char c;
        printf("Please type 1 character : ");
        c=getch();
        printf("You typed : %c\n", c);
        return 0;
}
```

- **puts**
  - outputs a string as a whole

```
#include <stdio.h>

int main()
{
        char a[80];
        puts("Type some words :");
        gets(a);
        puts(a);
        return 0;
}
```

- **printf**
  - to output a string

```
#include <stdio.h>

int main()
{
        char a[80]="abcd";
        printf("%s\n", a);
        return 0;
}
```

- **putchar**
  - outputs characters individually

```
#include <stdio.h>

int main()
{
        char letter;
        for (letter='A'; letter<='Z'; letter++)
                putchar (letter);
        printf("\n");
        return 0;
}
```

String Input and Output (cont.)

Program 9.1

```
#include <stdio.h>
int main()
{
  #define MSIZE 81
  char message[MSIZE]; /* enough storage for 80 characters plus '\0' */

  printf("Enter a string:\n");
  gets(message);
  printf("The string just entered is:\n");
  puts(message);

  return 0;
}
```

Sample run:

```
Enter a string:
This is a test input of a string of characters.
The string just entered is:
This is a test input of a string of characters.
```

String Input and Output (cont.)

- A **`printf()`** function call can be used in place of a **`puts()`** function call
  - `printf("%s\n",message);`
  - `puts(message);`
- This correspondence between the output functions is not duplicated by the input functions `scanf()` and `gets()`
  - `scanf()` reads a set of characters up to <u>either a blank space or a newline character</u>
    - `scanf("%s",message); //No & is required`
  - `gets()` stops accepting characters <u>only when a newline is detected</u>

String Processing

Element	String array	Expression	Value
Zero element	t	string2[0]!='\0'	1
First element	h	string2[1]!='\0'	1
Second element	i	string2[2]!='\0'	1
	s		
	i		
	s		
.		.	.
.	a	.	.
.		.	.
	s		
	t		
	r		
	i		
	n		
Fifteenth element	g	string2[15]!='\0'	1
Sixteenth element	\0	string2[16]!='\0'	0

End-of-string marker

Figure 9.3 The `while` test becomes false at the end of the string

NOTE:
The expression `string2[i]` is :

- non-0 for every other character
- only 0 at the end of a string

String Processing (cont.)

Program 9.3

getchar() is used to input a single character

```
#include <stdio.h>
int main()
{
  #define LSIZE 81
  char message[LSIZE];  /* enough storage for 80 characters plus '\0' */
  char c;
  int i;

  printf("Enter a string:\n");
  i = 0;
  while(i < (LSIZE-1) && (c = getchar()) != '\n')
  {
    message[i] = c; /* store the character entered */
    i++;
  }
  message[i] = '\0'; /* terminate the string */
  printf("The string just entered is: \n");
  puts(message);

  return 0;
}
```

Be careful: omitting the parentheses causes the entire expression to be equivalent to

```
c = (getchar() != '\n')
```

Library Functions

Table 9.2 String Library Routines (Required Header File is `string.h`)

Name	Description	Example
`strcpy(str1, str2)`	Copies `str2` to `str1`, including the `'\0'`	`strcpy(test, "efgh")`
`strcat(str1, str2)`	Appends `str2` to the end of `str1`	`strcat(test, "there")`
`strlen(string)`	Returns the length of `string`. Does not include the `'\0'` in the length count.	`strlen("Hello World!")`
`strcmp(str1, str2)`	Compares `str1` to `str2`. Returns a negative integer if `str1 < str2`, 0 if `str1 == str2`, and a positive integer if `str1 > str2`.	`strcmp("Beb", "Bee")`

Note: Attempting to copy a larger string into a smaller string causes the copy to overflow the destination array beginning with the memory area immediately following the last array element. (run-time error)

String Library Functions – string.h

- strcat (*string1*, *string2*)
  - concatenate string2 to string1
- strchr (*string*, *character*)
  - Locate the position of the first occurrence of character in string
- strcmp (*string1*, *string2*)
  - compare string2 to string1
- strcpy (*string1*, *string2*)
  - Make string1 equal to string2
- strlen (*string*)
  - Determine length of string

Sample of String Functions

word1	word2	strcat(word1,word2)
“alpha”	“beta”	word1 = “alphabeta” word2 = “beta”
word1	**word2**	**strchr(word1,word2)**
“alphabet”	‘a’	1

Sample of String Functions

word1	word2	strcmp(word1, word2)
“alpha”	“beta”	-1
“alpha”	“alpha”	1
word1	**word2**	**strcpy(word1, word2)**
“alpha”	“beta”	word1 = “beta” word2 = “beta”

word	strlen(word)
“alpha”	5
“alphabet”	8

EXERCISE:

1)

```
strcpy(s1,"Hello  ");
strcat(s1,"World”);
```

s1 will be ?

2)

```
char h[6] = "wild";
char p[6] = "crazy";
char s[10];
strcpy(s, h);
strcat(s, p);
```

s will be?

EXERCISE:

What will be displayed by the program below?

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

int main ( )
{
 char s1[9] = “jadavpur", s2[19] = “university";
char tmp1[10], tmp2[20];

strcpy(tmp2, s1);
strcat(tmp2, s2);
strcpy(tmp1, tmp2);
tmp1[8] = '\0';
printf("%s %s\n", tmp1, tmp2);
}
```

Library Functions (cont.)

Table 9.2 String Library Routines (Required Header File is `string.h`) (continued)

Name	Description	Example
`strncpy(str1, str2,n)`	Copies at most `n` characters of `str2` to `str1`. If `str2` has fewer than `n` characters, it pads `str1` with `'\0'`s.	`strncpy(str1, str2, 5)`
`strncmp(str1, str2,n)`	Compares at most `n` characters of `str1` to `str2`. Returns the same values as `strcmp()` based on the number of characters compared.	`strncmp("Beb", "Bee", 2)`
`strchr(string, char)`	Locates the position of the first occurrence of the **char** within `string`. Returns the address of the character.	`strchr("Hello", 'l')`
`strtok(string, char)`	Parses `string` into tokens. Returns the next sequence of **char** contained in `string` up to but not including the delimiter character.	`strtok("Hi Ho Ha", ' ')`

```
char a[10]="Bee", b[10]="Da";
strncpy(a, b, 2);
puts(a);
```

```
char a[10]="Bee",
b[10]="Beb";
int c;
c=strncmp(a, b, 2);
printf("%d\n", c);
```

```
char a[10]="Bee";
printf("%d\n", strchr(a, 'e'));
```

```
char str[100], limit[]= “”, *result =
NULL;
gets(str);
result = strtok( str, limit );
while( result != NULL )  {
printf("%s\n", result);
result = strtok( NULL, limit );
}
```

Library Functions (cont.)

Additional functions for strings: (required header file is string.h)

Name	Description	Example
strcmpi()	This function is same as strcmp() which compares 2 strings but not case sensitive.	strcmpi("THE","the"); will return 0.
strlwr()	This function converts all characters in a string from uppercase to lowercase.	strlwr("IIUM"); converts a string to "iium".
strupr()	This function converts all characters in a string from lower case to uppercase	strupr("knowledge"); converts a string to KNOWLEDGE
strrev()	This function reverses the characters in a string	strrev("program"); reverses a string into "margrop"

Library Functions (cont.)

- When comparing strings, their individual characters are evaluated in pairs; if a difference is found, the string with the first lower character is the smaller one
  - `"Good Bye"` is less than `"Hello"` because the first `'G'` in Good Bye is less than the first `'H'` in Hello
  - `"Hello"` is less than `"Hello "` because the `'\0'` terminating the first string is less than the `' '` in the second string
  - `"123"` is greater than `"122"` because `'3'` in `123` is greater than `'2'` in `122`
  - `"1237"` is greater than `"123"` because `'7'` in `1237` is greater than `'\0'` in `123`

Library Functions (cont.)

Program 9.5

```
1   #include <stdio.h>
2   #include <string.h> /* required for the string function library */
3
4   int main()
5   {
6     #define MAXELS 50
7     char string1[MAXELS] = "Hello";
8     char string2[MAXELS] = "Hello there";
9     int n;
10
11    n = strcmp(string1, string2);
12
13    if (n < 0)
14      printf("%s is less than %s\n\n", string1, string2);
15    else if (n == 0)
16      printf("%s is equal to %s\n\n", string1, string2);
17    else
18      printf("%s is greater than %s\n\n", string1, string2);
19
20    printf("The length of string1 is %d characters\n", strlen(string1));
```

n = -1 because “Hello” is less than “Hello there”

“Hello” = 5 characters

Library Functions (cont.)

```
21    printf("The length of string2 is %d characters\n\n", strlen(string2));
22
23    strcat(string1," there World!");
24
25    printf("After concatenation, string1 contains the string value\n");
26    printf("%s\n", string1);
27    printf("The length of this string is %d characters\n\n",
28                                                    strlen(string1));
29    printf("Type in a sequence of characters for string2:\n");
30    gets(string2);
31
32    strcpy(string1, string2);
33
34    printf("After copying string2 to string1");
35    printf(" the string value in string1 is:\n");
36    printf("%s\n", string1);
37    printf("The length of this string is %d characters\n\n",
38                                                    strlen(string1));
39    printf("\nThe starting address of the string1 string is: %d\n",
40                                                    (void *) string1);
41    return 0;
42  }
```

append " there World!" to string1

"Hello there" = 11 characters (including a blank space)

get a new string for string2

copy string2 to string1

String1 now contains "Hello there World!" = 18 characters

calculate number of characters in string1

Library Functions (cont.)

Sample output:

```
Hello is less than Hello there
The length of string1 is 5 characters
The length of string2 is 11 characters

After concatenation, string1 contains the string value
Hello there World!
The length of this string is 18 characters

Type in a sequence of characters for string2:
It's a wonderful day
After copying string2 to string1, the string value in
   string1 is:
It's a wonderful day
The length of this string is 20 characters

The starting address of the string1 string is: 1244836
```

The address depends on the compiler

String Programs Without Library Functions

- C Program to find the length of a string.

```
#include <stdio.h>
int main()
{
char s[20];
int i;
printf("Enter a string: ");
gets(s);
for(i = 0; s[i] != '\0'; ++i);
printf("Length of string: %d", i);
return 0;
}
```

String Programs Without Library Functions

- C Program to concatenate two strings.

```
#include <stdio.h>
int main()
{
char s1[100], s2[100], i, j;
printf("Enter first string: ");
gets(s1);
printf("Enter second string: ");
gets(s2);
// calculate the length of string s1
// and store it in i
for(i = 0; s1[i] != '\0'; ++i);
for(j = 0; s2[j] != '\0'; ++j, ++i)
{
s1[i] = s2[j];
}
s1[i] = '\0';
printf("After concatenation: %s", s1);
return 0;
}
```

String Programs Without Library Functions

- C Program to manually copy one string into another.

```
#include <stdio.h>
int main()
{
     char s1[100], s2[100], i;
    printf("Enter string s1: ");
    gets(s1);
  for(i = 0; s1[i] != '\0'; ++i)
     {
         s2[i] = s1[i];
      }
   s2[i] = '\0';
  printf("String s2: %s", s2);
return 0;
}
```

String Programs Without Library Functions

- C Program to find the frequency of a character in a string.

```
#include <stdio.h>
int main()
{
char str[1000], ch;
int i, frequency = 0;
printf("Enter a string: ");
gets(str);
printf("Enter a character to find the frequency: ");
scanf("%c",&ch);
    for(i = 0; str[i] != '\0'; ++i)
     {
        if(ch == str[i])
                ++frequency;
      }
printf("Frequency of %c = %d", ch, frequency);
return 0;
}
```

String Programs Without Library Functions

- C Program to convert a string to uppercase character.

```
#include <stdio.h>
#include <string.h>
int main()
{
  char s[100];   int i;
  printf("\nEnter a string : ");
  gets(s);
    for (i = 0; s[i]!='\0'; i++)
      {
         if(s[i] >= 'a' && s[i] <= 'z')
                  {            s[i] = s[i] - 32;
                   }
         }
    printf("\nString in Upper Case = %s", s);
 return 0;
}
```

String Programs Without Library Functions

- C Program to convert a string to lowercase character.

```
#include <stdio.h>
#include <string.h>
int main()
{
  char s[100];   int i;
  printf("\n Enter a string : ");
  gets(s);
    for (i = 0; s[i]!='\0'; i++)
      {
         if(s[i] >= 'A' && s[i] <= 'Z')
                  {            s[i] = s[i] + 32;
                   }
         }
    printf("\nString in Lower Case = %s", s);
 return 0;
}
```

String Programs Without Library Functions

- C Program to replace lowercase characters by uppercase characters and vice versa in a given string.

```c
#include <stdio.h>
#include <ctype.h>

 void main()
{
 char sentence[100];
 int count, ch, i;

printf("Enter a sentence \n");

for (i = 0;(sentence[i] = getchar()) != '\n'; i++);

  sentence[i] = '\0';  /*  shows the number of chars accepted in a sentence */

 count = i;

  printf("The given sentence is   : %s", sentence);

 printf("\n Case changed sentence is: ");

 for (i = 0; i < count; i++)
       {
          ch = islower(sentence[i])? toupper(sentence[i]) :tolower(sentence[i]);

        putchar(ch);
       }
}
```

String Programs Without Library Functions

- C Program to reverse the characters of a string.

```
#include<string.h>
#include<stdio.h>
 void  main()
{
    int  i,j,len=0;
    char  str[50],revstr[50];

    printf("\n Enter a String to Reverse : "  );
    gets(str);
       for(i=0; str[i]!='\0'; i++)
         {
            len++;
           }
        j=0;
            for(i=len-1; i>=0; i--)
                 {
                   revstr[j++]=str[i];
                  }
    printf("\n Reverse of the Given String is: %s",revstr);
    }
```

String Programs Without Library Functions

- C Program to reverse the words of a string.

```
#include <stdio.h>
int main(){

    char str[100],text[100];
    int i=0,j=0;
    printf("Enter Text:");
    gets(str);
            while(str[i]!='\0')    i++;
            while(i>0)      {
                            text[j]=str[--i];
                            ++j;
                            }
              text[j]='\0';
           printf("Reversed Text:");
                        for(i=0;text[i]!='\0';i++)
                              {
                                 if(text[i+1]==' ' || text[i+1]== NULL)     {

                                     for(j=i;j>=0 && text[j]!=' ';j--)
                                            printf("%c", text[j]);

                                              printf(" ");
                                                                            }
                                 }
        return 0;
    }
```

String Programs Without Library Functions

- C Program to count number of vowels, consonants, spaces and digits in a given string.

```
#include <stdio.h>
int main()
{
char line[150];
int i, vowels=0, consonants=0, digits=0, spaces=0;
printf("Enter a line of string: ");
scanf("%[^\n]", line);
for(i=0; line[i]!='\0'; ++i)
{
     if(line[i]=='a' || line[i]=='e' || line[i]=='i' || line[i]=='o' || line[i]=='u' || line[i]=='A' || line[i]=='E' || line[i]=='I' ||
line[i]=='O' || line[i]=='U')
          {
            ++vowels;
          }
    else if((line[i]>='a'&& line[i]<='z') || (line[i]>='A'&& line[i]<='Z'))
         {
            ++consonants;
          }
     else if(line[i]>='0' && line[i]<='9')
         {
             ++digits;
         }
     else if (line[i]==' ')
        {
            ++spaces;
        }
}
printf("Vowels: %d",vowels);
printf("\nConsonants: %d",consonants);
printf("\nDigits: %d",digits);
printf("\nWhite spaces: %d", spaces);
return 0; }
```

String Programs Without Library Functions

- C Program to remove all the special characters (except alphabets) from a given string.

```
#include<stdio.h>
int main()
{
char line[150];
int i, j;
printf("Enter a string: ");
gets(line);
     for(i = 0; line[i] != '\0'; ++i)
   {
  while (!( (line[i] >= 'a' && line[i] <= 'z') || (line[i] >= 'A' && line[i] <= 'Z') || line[i] == '\0') )
            {
               for(j = i; line[j] != '\0'; ++j)
                     {
                        line[j] = line[j+1];
                       }
                line[j] = '\0';
             }
       }
printf("Output String: ");
puts(line);
return 0;
}
```

String Programs Without Library Functions

- C Program to check whether a given string is an anagram or not.

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

int main () {
   char s1[] = "recitals";
   char s2[] = "articles";
    char temp;
    int i, j;
   int n  = strlen(s1);
   int n1 = strlen(s2);

    // If both strings are of different length, then they are not
anagrams

    if( n != n1) {
      printf("%s and %s are not anagrams! \n", s1, s2);
      return 0;
   }
      // lets sort both strings first -

   for (i = 0; i < n-1; i++) {
      for (j = i+1; j < n; j++) {
         if (s1[i] > s1[j]) {
            temp  = s1[i];
            s1[i] = s1[j];
            s1[j] = temp;
         }
```

```
if (s2[i] > s2[j]) {
            temp  = s2[i];
            s2[i] = s2[j];
            s2[j] = temp;
        }
    }
  }

// Compare both strings character by character

  for(i = 0; i<n; i++)
   {
     if(s1[i] != s2[i])
        {
       printf("Strings %s and %s are not anagrams! \n", s1,s2);
         }
      else
        printf("Strings %s and %s are anagrams! \n", s1,s2);
    }
  return 0;
}
```

Character Routines (cont.)

Table 9.3 Character Library Routines (Required Header File is `ctype.h`)

Required Prototype	Description	Example
`int isalpha(char)`	Returns a non-0 number if the character is a letter; otherwise, it returns 0.	`isalpha('a')`
`int isupper(char)`	Returns a non-0 number if the character is uppercase; otherwise, it returns 0.	`isupper('a')`
`int islower(char)`	Returns a non-0 number if the character is lowercase; otherwise, it returns 0.	`islower('a')`
`int isdigit(char)`	Returns a non-0 number if the character is a digit (0 through 9); otherwise, it returns 0.	`isdigit('a')`
`int isascii(char)`	Returns a non-0 number if the character is an ASCII character; otherwise, it returns 0.	`isascii('a')`
`int isspace(char)`	Returns a non-0 number if the character is a space; otherwise, it returns 0.	`isspace(' ')`
`int isprint(char)`	Returns a non-0 number if the character is a printable character; otherwise, it returns 0.	`isprint('a')`
`int iscntrl(char)`	Returns a non-0 number if the character is a control character; otherwise, it returns 0.	`iscntrl('a')`
`int ispunct(char)`	Returns a non-0 number if the character is a punctuation character; otherwise, it returns 0.	`ispunct('!')`
`int toupper(char)`	Returns the uppercase equivalent if the character is lowercase; otherwise, it returns the character unchanged.	`toupper('a')`
`int tolower(char)`	Returns the lowercase equivalent if the character is uppercase; otherwise, it returns the character unchanged.	`tolower('A')`

Character Routines (cont.)

- **isalpha** returns true if the character is in the range of A-Z or a-z.
- **isdigit** returns true if the character is in the range of 0-9.
- **islower** returns true if the character is in the range of a-z.
- **isupper** returns true if the character is in the range of A-Z.
- **tolower** if isupper return the lowercase character otherwise return the character.
- **toupper** if islower return the uppercase character otherwise return the character.

Character Routines (cont.)

- isalpha
  - isalpha (“s”) returns 1
  - isalpha (“4”) returns 0
- isupper
  - isupper (“S”) returns 1
  - isupper (“s”) returns 0
- islower (*character*)
  - islower (“S”) returns 0
  - islower (“s”) returns 1
- isdigit (*character*)
  - isalpha (“s”) returns 0
  - isalpha (“4”) returns 1

- toupper (character)
  - toupper (“g”) returns “G”
  - toupper (“G”) returns “G”
- tolower (character)
  - tolower (“Q”) returns “q”
  - tolower (“q”) returns “q”

Character Routines (cont.)

Program 9.6

```
1   #include <stdio.h>
2   #include <ctype.h> /* required for the character function library */
3
4   int main()
5   {
6     #define MAXCHARS 100
7     char message[MAXCHARS];
8     void convertToUpper(char []);  /* function prototype */
9
10    printf("\nType in any sequence of characters:\n");
11    gets(message);
12
13    convertToUpper(message);
14
15    printf("The characters just entered, in uppercase are:\n%s\n", message);
16
17    return 0;
18  }
19  // this function converts all lowercase characters to uppercase
20  void convertToUpper(char message[])
21  {
22    int i;
23    for(i = 0; message[i] != '\0'; i++)
24      message[i] = toupper(message[i]);
25  }
```

A function call (passing a string to a function called convertToUpper()

Formatting Strings

- Examples: (By using field width specifier)

  - `printf("|%25s|","Have a Happy Day");`

  (Displays the message right-justified in a field of 25 char)

  Output: |^^^^^^^^^^Have a Happy Day|

  - `printf("|%-25s|","Have a Happy Day");`

  (Displays the message left-justified in a field of 25 char)

  Output: |Have a Happy Day^^^^^^^^^^|

  - `printf("|%.12s|","Have a Happy Day");`

  (Displays the first 12 chararacters in the string to be displayed)

  Output: |Have a Happy|

  - `printf("|%25.12s|","Have a Happy Day");`

  (Displays the message right-justified in a field of 25 char and first 12 chararacters in the string to be displayed)

  Output: |^^^^^^^^^^^^^Have a Happy|

^ to denote a blank space

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”};

Sort Strings in Dictionary Order

```
#include <stdio.h>
#include <string.h>
int main() {
char str[5][50], temp[50];
printf("Enter 5 words: ");
        for(int i = 0; i < 5; ++i) {
                fgets(str[i], sizeof(str[i]), stdin);
                }
    for(int i = 0; i < 5; ++i) {
    for(int j = i+1; j < 5 ; ++j) {
        if(strcmp(str[i], str[j]) > 0) {
                        strcpy(temp, str[i]);
                        strcpy(str[i], str[j]);
                        strcpy(str[j], temp);
                }
            }
        }
printf("\nIn the lexicographical order: \n");
        for(int i = 0; i < 5; ++i) {
            fputs(str[i], stdout);
            }
return 0;
}
```