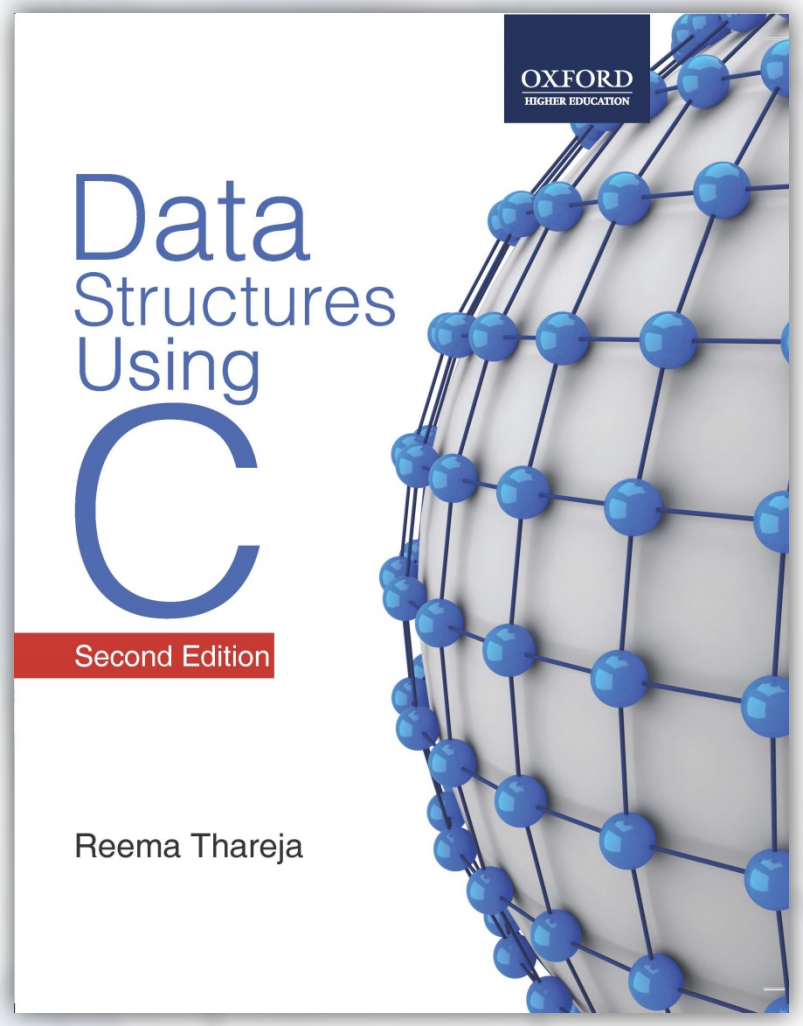

Data Structures Using C, 2e

Reema Thareja

Chapter 4

Strings

Introduction

- A string is a null-terminated character array. This means that after the last character, a null character (‘\0’) is stored to signify the end of the character array.
- The general form of declaring a string is

  char str[size];

- For example if we write,

  char str[] = “HELLO”;

Introduction

- We are declaring a character array with 5 characters namely, H, E, L, L and O. Besides, a null character (‘\0’) is stored at the end of the string. So, the internal representation of the string becomes HELLO‘\0’.
- Note that to store a string of length 5, we need 5 + 1 locations (1 extra for the null character).
- The name of the character array (or the string) is a pointer to the beginning of the string.

Reading Strings

If we declare a string by writing

```
char str[100];
```

Then str can be read from the user by using three ways

- using scanf function
- using gets() function
- using getchar() function repeatedly

str can be read using scanf() by writing

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

str can be read by writing

```
gets(str);
```

gets() takes the starting address of the string which will hold the input. The string inputted using gets() is automatically terminated with a null character.

Reading Strings

- str can also be read by calling the getchar() function repeatedly to read a sequence of single characters (unless a terminating character is entered) and simultaneously storing it in a character array.

```
i=0;
ch = getchar ();
while(ch != '*’)
{    str[i] = ch;
     i++;
     ch = getchar;
}    str[i] = '\0';
```

Writing Strings

- Strings can be displayed on screen using three ways

  using printf() function

  using puts() function

  using putchar() function repeatedly

- str can be displayed using printf() by writing

  printf("%s", str);

- str can be displayed by writing

  puts(str);

Writing Strings

str can also be written by calling the putchar() repeatedly to print a sequence of single characters

```
i=0;
while(str[i] != '\0’)
{    putchar(str[i]);
     i++;
}
```

Finding Length of a String

- The number of characters in a string constitutes the length of the string.
- For example, LENGTH("C PROGRAMMING IS FUN") will return 20.

  Note that even blank spaces are counted as characters in the string.

```
ALGORITHM TO CALCULATE THE LENGTH OF A STRING
Step 1: [INITIALIZE] SET I = 0
Step 2: Repeat Step 3 while STR[I] != NULL
Step 3:         SET I = I + 1
        [END OF LOOP]
Step 4: SET LENGTH = I
Step 5: END
```

Converting Characters of a String into Upper Case

- In memory the ASCII code of a character is stored instead of its real value.
- The ASCII code for A-Z varies from 65 to 91 and the ASCII code for a-z ranges from 97 to 123.
- So if we have to convert a lower case character into upper case, then we just need to subtract 32 from the ASCII value of the character.

Converting Characters of a String into Upper Case

```
ALGORITHM TO CONVERT THE CHARACTERS OF STRING INTO UPPER CASE

Step1:  [Initialize] SET I=0
Step 2: Repeat Step 3 while STR[I] != NULL
Step 3:        IF STR[1] >= 'a' AND STR[I] <= 'z'
               SET Upperstr[I] = STR[I] - 32
          ELSE
               SET Upperstr[I] = STR[I]
              [END OF IF]
        [END OF LOOP]
Step 4: SET Upperstr[I] = NULL
Step 5: EXIT
```

Appending a String to Another String

- Appending one string to another string involves copying the contents of the source string at the end of the destination string.
- For example, if S1 and S2 are two strings, then appending S1 to S2 means we have to add the contents of S1 to S2.
- So S1 is the source string and S2 is the destination string.
- The appending operation would leave the source string S1 unchanged and destination string S2 = S2+S1.

Appending a String to Another String

```
ALGORITHM TO APPEND A STRING TO ANOTHER STRING

    Step 1: [Initialize] SET I =0 and J=0
    Step 2: Repeat Step 3 while Dest_Str[I] != NULL
    Step 3:  SET I + I + 1
            [END OF LOOP]
    Step 4: Repeat Steps 5 to 7 while Source_Str[J] != NULL
    Step 5:  Dest_Str[I] = Source_Str[J]
    Step 6:  SET I = I + 1
    Step 7:  SET J = J + 1
            END OF LOOP]
    Step 8: SET Dest_Str[I] = NULL
    Step 9: EXIT
```

Comparing Two Strings

- If S1 and S2 are two strings then comparing two strings will give either of these results:

✔ S1 and S2 are equal

✔ S1>S2, when in dictionary order S1 will come after S2

✔ S1<S2, when in dictionary order S1 precedes S2

Comparing Two Strings

```
Step1: [Initialize] SET I=0, SAME =0
Step 2: SET Len1 = Length(STR1), Len2 = Length(STR2)
Step 3: IF len1 != len2, then
          Write "Strings Are Not Equal"
        ELSE
          Repeat while I<Len1
               IF STR1[I] == STR2[I]
                    SET I = I + 1
               ELSE
                    Go to Step 4
               [END OF IF]
          [END OF LOOP]
          IF I = Len1, then
               SET SAME =1
               Write "Strings are equal"
                    [END OF IF]
Step 4:  IF SAME = 0, then
          IF STR1[I] > STR2[I], then
               Write "String1 is greater than String2"
                    ELSE IF STR1[I] < STR2[I], then
               Write "String2 is greater than String1"
          [END OF IF]
         [END OF IF]
Step 5: EXIT
```

Reversing a String

- If S1= “HELLO”, then reverse of S1 = “OLLEH”.
- To reverse a string we just need to swap the first character with the last, second character with the second last character, so on and so forth.

```
ALGORITHM TO REVERSE A STRING

Step1: [Initialize] SET I=0, J= Length(STR)-1
Step 2: Repeat Steps 3 and 4 while I< J
Step 3:   SWAP( STR(I), STR(J))
Step 4:   SET I = I + 1, J = J - 1
       [END OF LOOP]
Step 5: EXIT
```

Extracting a Substring from a String

- To extract a substring from a given string requires information about three things

✔ the main string

✔ the position of the first character of the substring in the given string

✔ maximum number of characters/length of the substring

```
Algorithm to extract substring from a given text

     Step 1: [INITIALIZE] Set I=M, J = 0
     Step 2: Repeat Steps 3 to 6 while str[I] != NULL and N>0
     Step 3:   SET substr[J] = str[I]
     Step 4:   SET I = I + 1
     Step 5:   SET J = J + 1
     Step 6:   SET N = N - 1
             [END OF LOOP]
     Step 7: SET substr[J] = NULL
     Step 8: EXIT
```

Inserting a String in the Main String

- The insertion operation inserts a string S in the main text, T at the kth position.

```
Algorithm to insert a string in the main text

     Step 1: [INITIALIZE] SET I=0, J=0 and K=0
     Step 2: Repeat Steps 3 to 4 while text[I] != NULL
     Step 3: IF I = pos, then
               Repeat while str[K] != NULL
               new_str[j] = str[k]
               SET J=J+1
               SET K = K+1
               [END OF INNER LOOP]
             ELSE
              new_str[[J] = text[I]
             SET J = J+1
             [END OF IF]
     Step 4: SET I = I+1
           [END OF OUTER LOOP]
     Step 5: SET new_str[J] = NULL
     Step 6: EXIT
```

Pattern Matching

- This operation returns the position in the string where the string pattern first occurs. For example,
- INDEX (“Welcome to the world of programming”, “world”) = 15
- However, if the pattern does not exist in the string, the INDEX function returns 0.

```
Algorithm to find the index of the first occurrence of a string within a
given text

     Step 1: [Initialize] SET I=0 and MAX = LENGTH(text) - LENGTH(str) +1
     Step 2: Repeat Steps 3 to 6 while I <= MAX
     Step 3:   Repeat step 4 for K = 0 To Length(str)
     Step 4:        IF str[K] != text[I + K], then GOTO step 6
                    [END of INNER LOOP]
     Step 5:   SET INDEX = I. Goto step 8
     Step 6:   SET I = I+1
              [END OF OUTER LOOP]
     Step 7: SET INDEX = -1
     Step 8: EXIT
```

Deleting a Substring from a Main String

- The deletion operation deletes a substring from a given text. We write it as, DELETE(text, position, length)
- For example, DELETE("ABCDXXXABCD", 5, 3) = "ABCDABCD"

```
Algorithm to delete a substring from a text

     Step 1: [INITIALIZE] SET I=0 and J=0
     Step 2: Repeat steps 3 to 6 while text[I] != NULL
     Step 3:   IF I=M, then
                        Repeat while N>=0
                    SET I = I+1
                    SET N = N - 1
                        [END OF INNER LOOP]
                   [END OF IF]
     Step 4: SET new_str[J] = text[I]
     Step 5: SET J= J + 1
     Step 6: SET I = I + 1
             [END OF OUTER LOOP]
     Step 7: SET new_str[J] = NULL
     Step 8: EXIT
```

Replacing a Pattern with Another Pattern in a String

- Replacement operation is used to replace the pattern P1 by another pattern P2. This is done by writing, REPLACE (text, pattern1, pattern2)
- For example, (“AAABBBCCC”, “BBB”, “X”) = AAAXCCC

  (“AAABBBCCC”, “X”, “YYY”)= AAABBBCC

```
Algorithm to replace a pattern P1 with another pattern P2 in the given text
TEXT

     Step 1: [INITIALIZE] SET Pos = INDEX(TEXT, P1)
     Step 2: SET TEXT = DELETE(TEXT, Pos, LENGTH(P1))
     Step 3: INSERT(TEXT, Pos, P2)
     Step 4: EXIT
```

Arrays of Strings

- Suppose there are 20 students in a class and we need a string that stores names of all the 20 students. How can this be done? Here, we need a string of strings or an array of strings. Such an array of strings would store 20 individual strings.
- An array of strings is declared as: char names[20][30];
- Here, the first index will specify how many strings are needed and the second index specifies the length of every individual string. So we allocate space for 20 names where each name can be maximum 30 characters long.

Arrays of Strings

Let us see the memory representation of an array of strings.

If we have an array declared as,
char name[5][10] = {“Ram”, “Mohan”, “Shyam”, “Hari”, “Gopal”};

Name[0]	R	A	M	‘\0’					
Name[1]	M	O	H	A	N	‘\0’			
Name[2]	S	H	Y	A	M	‘\0’			
Name[3]	H	A	R	I	‘\0]				
Name[4]	G	O	P	A	L	‘\0’			

Pointers and Strings

Now, consider the following program that prints a text.

```
#include<stdio.h>
main()
{     char str[] = “Oxford”;
            char *pstr = str;
      printf(“\n The string is : ”);
      while( *pstr != ‘\0’)
      {     printf(“%c’, *pstr);
            pstr++;
      }
}
```

Pointers and Strings

- In this program we declare a character pointer *pstr to show the string on the screen.
- We then "point" the pointer pstr at str.
- Then we print each character of the string in the while loop.
- Instead of using the while loop, we could have straight away used the puts() function, like puts(pstr);