BLM267

Chapter 4: Strings

Data Structures Using C, Second Edition
Reema Thareja

- Introduction
- Operations on Strings
- Arrays of Strings
- Pointers and Strings

Introduction

- Computers are widely used for word processing applications such as creating, inserting, updating, and modifying textual data.
- In C, 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.
- For example, if we write char str[] = "HELLO"; then we are declaring an array that has five characters, namely, H, E, L, L, and O.
- Apart from these characters, a null character ('\0') is stored at the end of the string.
- So, the internal representation of the string becomes HELLO'\0'. 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.

Introduction

- Like we use subscripts (also known as index) to access the elements of an array, we can also use subscripts to access the elements of a string.
- The subscript starts with a zero (0). All the characters of a string are stored in successive memory locations.
- Figure 4.2 shows how str[] is stored in the memory. Thus, in simple terms, a string is a sequence of characters.
- For simplicity, the figure shows that H is stored at memory location 1000 but in reality, the ASCII code of a character is stored in the memory and not the character itself.
- So, at address 1000, 72 will be stored as the ASCII code for H is 72.

str[0]	1000	H
str[1]	1001	E
str[2]	1002	L
str[3]	1003	L
str[4]	1004	O
str[5]	1005	\0

Figure 4.2 Memory representation of a character array

Programming Tip

When allocating memory space for a string, reserve space to hold the null character also.

Introduction

- The statement **char str[] = "HELLO";** declares a constant string, as we have assigned a value to it while declaring the string.
- The general form of declaring a string is char str[size];
- When we declare the string like this, we can store size–1 characters in the array because the last character would be the null character.
  - For example, char m[100]; can store a maximum of 99 characters.
- The other way to initialize a string is to initialize it as an array of characters.
- For example, char str[] = {'H', 'E', 'L', 'L', 'O', '\0'}; In this example, we have explicitly added the null character.

Introduction

- We can also declare a string with size much larger than the number of elements that are initialized. For example, consider the statement below. char str [10] = "HELLO";
- In such cases, the compiler creates an array of size 10; stores "HELLO" in it and finally terminates the string with a null character. Rest of the elements in the array are automatically initialized to NULL.
- Now consider the following statements: char str[3]; str = "HELLO";
- The above initialization statement is illegal in C and would generate a compile-time error because of two reasons.
- First, the array is initialized with more elements than it can store.
- Second, initialization cannot be separated from declaration.

Introduction

Reading Strings

- If we declare a string by writing char str[100];
- Then str can be read by the user in three ways:
  1. using scanf function,
  2. using gets() function, and
  3. using getchar() function repeatedly.
- Strings can be read using scanf() by writing scanf("%s", str);
- scanf() function is well known and easy to use,
  - the main pitfall of using this function is that the function terminates as soon as it finds a blank space.
- For example, if the user enters Hello World, then the str will contain only Hello.

Introduction

Reading Strings

- The next method of reading a string is by using the gets() function.
- The string can be read by writing gets(str); gets() is a simple function that overcomes the drawbacks of the scanf() function.
- The gets() function takes the starting address of the string which will hold the input.
- The string inputted using gets() is automatically terminated with a null character.

Introduction

- Reading Strings
- Strings 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 as shown below.

```
i=0;
ch = getchar();// Get a character
while(ch != '*')
{
        str[i] = ch;// Store the read character in str
        i++;
        ch = getchar();// Get another character
}
str[i] = '\0';// Terminate str with null character
```

- Note that in this method, you have to deliberately append the string with a null character.
- The other two functions automatically do this.

Introduction

- **Writing Strings**
- Strings can be displayed on the screen using the following three ways:
  1. using printf() function,
  2. using puts() function, and
  3. using putchar() function repeatedly.
- Strings can be displayed using printf() by writing

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

- The precision specifies the maximum number of characters to be displayed, after which the string is truncated. The below statement would print only the first three characters in a total field of five characters.

```
printf ("%5.3s", str);
```

- Also these characters would be right justified in the allocated width.
- To make the string left justified, we must use a minus sign.

```
printf ("%–5.3s", str);
```

Introduction

- The next method of writing a string is by using puts() function.
  - A string can be displayed by writing puts(str);

- Strings can also be written by calling the putchar() function repeatedly to print a sequence of single characters.

```
i=0;
while(str[i] != '\0')
 {
   putchar(str[i]);
   // Print the character on the screen
   i++;
 }
```

Operations on Strings

- **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.
- The library function strlen(s1) which is defined in string.h returns the length of string s1.

Operations on Strings

- Figure 4.3 shows an algorithm that calculates the length of a string.
  - In this algorithm, I is used as an index for traversing string STR.
  - To traverse each and every character of STR, we increment the value of I.
  - Once we encounter the null character, the control jumps out of the while loop and the length is initialized with the value of I.
  - Note The library function strlen(s1) which is defined in string.h returns the length of string s1.

```
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
```

Figure 4.3 Algorithm to calculate the length of a string

Operations on Strings

1. Write a program to find the length of a string.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        char str[100], i = 0, length;
        clrscr();
        printf("\n Enter the string : ");
        gets(str)
        while(str[i] != '\0')
                i++;
        length = i;
        printf("\n The length of the string is : %d", length);
        getch()
        return 0;
}
```

Output

```
Enter the string : HELLO
The length of the string is : 5
```

Operations on Strings

- Converting Characters of a String into Upper/ Lower Case
- 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 uppercase, we just need to subtract 32 from the ASCII value of the character.
- And if we have to convert an upper case character into lower case, we need to add 32 to the ASCII value of the character.
- The library functions toupper() and tolower() which are defined in ctype.h convert a character into upper and lower case, respectively.

Operations on Strings

- Figure 4.4 shows an algorithm that converts the lower case characters of a string into upper case.
  - Using I as the index of STR, we traverse each character of STR from Step 2 to 3.
  - If the character is in lower case, then it is converted into upper case by subtracting 32 from its ASCII value.
  - But if the character is already in upper case, then it is copied into the UPPERSTR string.
  - Finally, when all the characters have been traversed, a null character is appended to UPPERSTR (as done in Step 4).

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

Figure 4.4 Algorithm to convert characters of a string into upper case

Operations on Strings

2. Write a program to convert the lower case characters of a string into upper case.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        char str[100], upper_str[100];
        int i=0;
        clrscr();
        printf("\n Enter the string :");
        gets(str);
        while(str[i] != '\0')
        {
                if(str[i]>='a' && str[i]<='z')
                        upper_str[i] = str[i] - 32;
                else
                        upper_str[i] = str[i];
                i++;
        }
        upper_str[i] = '\0';
        printf("\n The string converted into upper case is : ");
        puts(upper_str);
        return 0;
}
```

Output

```
Enter the string : Hello
The string converted into upper case is : HELLO
```

Operations on Strings

- 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.
  - S1 is the source string and S2 is the destination string.
  - The appending operation would leave the source string S1 unchanged and the destination string S2 = S2 + S1.
- Note The library function strcat(s1, s2) which is defined in string.h concatenates string s2 to s1.

Operations on Strings

- Figure 4.5 shows an algorithm that appends two strings.
  - In this algorithm, we first traverse through the destination string to reach its end, that is, reach the position where a null character is encountered.
  - The characters of the source string are then copied into the destination string starting from that position.
  - Finally, a null character is added to terminate the destination 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
```

Figure 4.5 Algorithm to append a string to another string

Operations on Strings

3. Write a program to append a string to another string.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        char Dest_Str[100], Source_Str[50];
        int i=0, j=0;
        clrscr();
        printf("\n Enter the source string : ");
        gets(Source_Str);
        printf("\n Enter the destination string : ");
        gets(Dest_Str);
        while(Dest_Str[i] != '\0')
                i++;
        while(Source_Str[j] != '\0')
        {
                Dest_Str[i] = Source_Str[j];
                i++;
                j++;
        }
        Dest_Str[i] = '\0';
        printf("\n After appending, the destination string is : ");
        puts(Dest_Str);
        getch();
        return 0;
}
```

Output

```
Enter the source string : How are you?
Enter the destination string : Hello,
After appending, the destination string is : Hello, How are you?
```

Operations on Strings

- Comparing Two Strings
- If S1 and S2 are two strings, then comparing the two strings will give either of the following results:
  - (a) S1 and S2 are equal
  - (b) S1>S2, when in dictionary order, S1 will come after S2
  - (c) S1<S2, when in dictionary order, S1 precedes S2
- To compare the two strings, each and every character is compared from both the strings.
- If all the characters are the same, then the two strings are said to be equal.
- Note The library function strcmp(s1, s2) which is defined in string.h compares string s1 with s2.

Operations on Strings

- Figure 4.6 shows an algorithm that compares two strings.
  - We first check whether the two strings are of the same length.
    - If not, then there is no point in moving ahead.

```
Step 1: [INITIALIZE] SET I=0, SAME =0
Step 2: SET LEN1 = Length(STR1), LEN2 = Length(STR2)
Step 3: IF LEN1 != LEN2
            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
                  SET SAME =1
                  Write "Strings are Equal"
            [END OF IF]
Step 4: IF SAME = 0,
            IF STR1[I] > STR2[I]
                  Write "String1 is greater than String2"
            ELSE IF STR1[I] < STR2[I]
                  Write "String2 is greater than String1"
            [END OF IF]
        [END OF IF]
Step 5: EXIT
```

Figure 4.6 Algorithm to compare two strings

- If the two strings are of the same length, then we compare character by character to check if all the characters are same.
- If yes, then the variable SAME is set to 1.
- Else, if SAME = 0, then we check which string precedes the other in the dictionary order

Operations on Strings

- 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, and so on.
- Note The library function strrev(s1) which is defined in string.h reverses all the characters in the string except the null character.

Operations on Strings

- Figure 4.7 shows an algorithm that reverses a string.
  - In Step 1, I is initialized to zero and J is initialized to the length of the string –1.
  - In Step 2, a while loop is executed until all the characters of the string are accessed.
  - In Step 3, we swap the ith character of STR with its jth character.
  - As a result, the first character of STR will be replaced with its last character, the second character will be replaced with the second last character of STR, and so on.
  - In Step 4, the value of I is incremented and J is decremented to traverse STR in the forward and backward directions, respectively.

```
Step 1: [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
```

Figure 4.7 Algorithm to reverse a string

Operations on Strings

5. Write a program to reverse a given string.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main()
{
        char str[100], reverse_str[100], temp;
        int i=0, j=0;
        clrscr();
        printf("\n Enter the string : ");
        gets(str);
        j=strlen(str)-1;
        while(i<j)
        {
                temp = str[j];

              str[j] = str[i];
              str[i] = temp;
              i++;
              j--;
        }
        printf("\n The reversed string is : ");
        puts(str);
        getch();
        return 0;
}
```

Output

```
Enter the string: Hi there
The reversed string is: ereht iH
```

Operations on Strings

- **Extracting a Substring from a String**
- To extract a substring from a given string, we need the following three parameters:

    1. the main string,

    2. the position of the first character of the substring in the given string, and

    3. the maximum number of characters/length of the substring.
- For example, if we have a string str[] = "Welcome to the world of programming";
- Then, SUBSTRING(str, 15, 5) = world

Operations on Strings

- Figure 4.8 shows an algorithm that extracts a substring from the middle of a string.
  - In this algorithm, we initialize a loop counter I to M, that is, the position from which the characters have to be copied.
  - Steps 3 to 6 are repeated until N characters have been copied.
  - With every character copied, we decrement the value of N.
  - The characters of the string are copied into another string called the SUBSTR.
  - At the end, a null character is appended to SUBSTR to terminate the string.

```
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
```

Figure 4.8 Algorithm to extract a substring from the middle of a string

Operations on Strings

- **Inserting a String in the Main String**
- The insertion operation inserts a string S in the main text T at the kth position.
- The general syntax of this operation is

  INSERT(text, position, string).
- For example,

  INSERT("XYZXYZ", 3, "AAA") = "XYZAAAXYZ"

Operations on Strings

- Figure 4.9 shows an algorithm to insert a string in a given text at the specified position.
  - This algorithm first initializes the indices into the string to zero.
  - From Steps 3 to 5, the contents of NEW_STR are built.
  - If I is exactly equal to the position at which the substring has to be inserted, then the inner loop copies the contents of the substring into NEW_STR.
  - Otherwise, the contents of the text are copied into it.

```
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
          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
```

Figure 4.9 Algorithm to insert a string in a given text at the specified position

Operations on Strings

- 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

Operations on Strings

- Figure 4.10 shows an algorithm to find the index of the first occurrence of a string within a given text.
  - In this algorithm, MAX is initialized to length(TEXT) – Length(STR) + 1.
  - For example, if a text contains 'Welcome To Programming' and the string contains 'World', in the main text, we will look for at the most 22 – 5 + 1 = 18 characters because after that there is no scope left for the string to be present in the text.
  - Steps 3 to 6 are repeated until each and every character of the text has been checked for the occurrence of the string within it.
  - In the inner loop in Step 3, we check the n characters of string with the n characters of text to find if the characters are same.
  - If it is not the case, then we move to Step 6, where I is incremented.
  - If the string is found, then the index is initialized with I, else it is set to –1.

```
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
```

Figure 4.10 Algorithm to find the index of the first occurrence of a string within a given text

Operations on Strings

- Deleting a Substring from the Main String
- The deletion operation deletes a substring from a given text.
  - DELETE(text, position, length).
  - For example,
    - DELETE("ABCDXXXABCD", 4, 3) = "ABCDABCD"

Operations on Strings

- **Figure 4.11 shows an algorithm to delete a substring from a given text. In this algorithm, we first initialize the indices to zero.**
  - **Steps 3 to 6 are repeated until all the characters of the text are scanned.**
  - **If I is exactly equal to M (the position from which deletion has to be done), then the index of the text is incremented and N is decremented.**
  - **N is the number of characters that have to be deleted starting from position M.**
  - **However, if I is not equal to M, then the characters of the text are simply copied into the NEW_STR.**

```
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
               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
```

Figure 4.11 Algorithm to delete a substring from a text

Operations on Strings

- Replacing a Pattern with Another Pattern in a String
- The replacement operation is used to replace the pattern P1 by another pattern P2.
  - REPLACE(text, pattern1, pattern2).
  - For example, ("AAABBBCCC", "BBB", "X") = AAAXCCC
- Figure 4.12 shows an algorithm to replace a pattern P1 with another pattern P2 in the text.
- The algorithm is very simple, where we first find the position POS, at which the pattern occurs in the text, then delete the existing pattern from that position and insert a new pattern there.

```
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
```

Figure 4.12 Algorithm to replace a pattern P_1 with another pattern P_2 in the text

Arrays of Strings

- Till now we have seen that a string is an array of characters.
- For example, if we say char name[] = "Mohan", then the name is a string (character array) that has five characters.
- Now, suppose that there are 20 students in a class and we need a string that stores the names of all the 20 students.
- Such an array of strings would store 20 individual strings.
  - char names[20][30];
- Here, the first index will specify how many strings are needed and the second index will specify the length of every individual string.

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"};
- Then in the memory, the array will be stored as shown in Fig. 4.13.
- By declaring the array names, we allocate 50 bytes.
- But the actual memory occupied is 27 bytes.
- Thus, we see that about half of the memory allocated is wasted.

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'				

Figure 4.13 Memory representation of a 2D character array

Pointers and Strings

- Now, consider the following program that prints a text.

```
#include <stdio.h>
int main()
{
 char str[] = "Hello";
 char *pstr;
 pstr = str;
 printf("\n The string is : ");
 while(*pstr != '\0')
 {
         printf("%c", *pstr);
         pstr++;
 }
 return 0;
}
```

Output The string is: Hello

Pointers and Strings

```
#include <stdio.h>
int main()
{
        char str[100], *pstr;
        int upper = 0, lower = 0;
        printf("\n Enter the string : ");
        gets(str);
        pstr = str;
        while(*pstr != '\0')
        {
                if(*pstr >= 'A' && *pstr <= 'Z')
                        upper++;
                else if(*pstr >= 'a' && *pstr <= 'z')
                        lower++;
                pstr++;
        }
        printf("\n Total number of upper case characters = %d", upper);
        printf("\n Total number of lower case characters = %d", lower);
        return 0;
}
```

Output

```
Enter the string : How are you
Total number of upper case characters = 1
Total number of lower case characters = 8
```