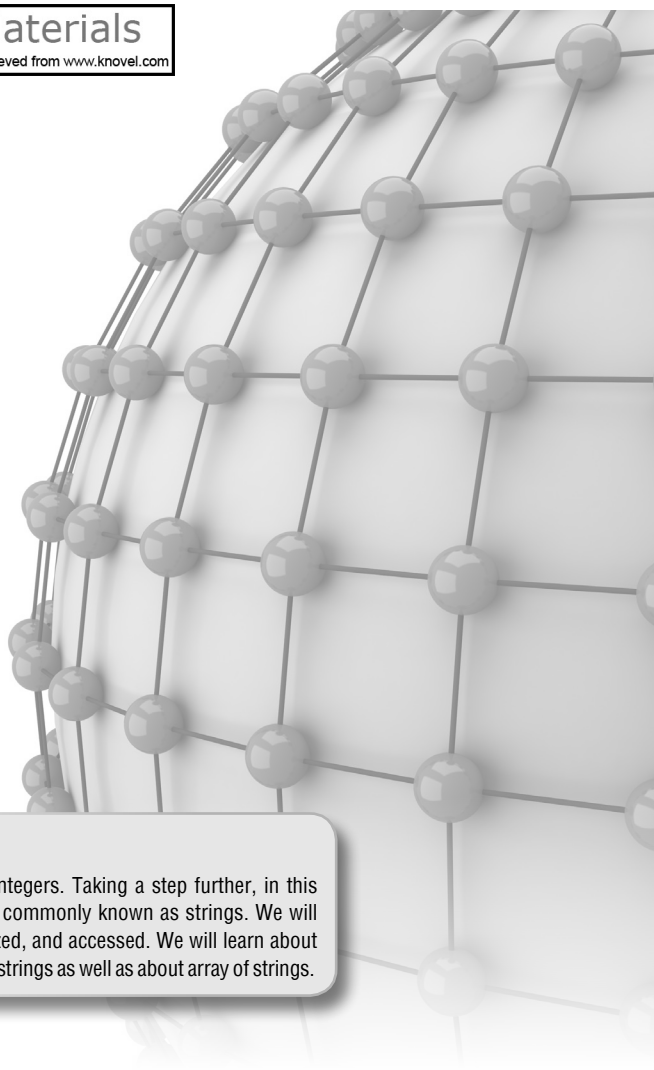
CHAPTER 4

Strings



LEARNING OBJECTIVE

In the last chapter, we discussed array of integers. Taking a step further, in this chapter, we will discuss array of characters commonly known as strings. We will see how strings are stored, declared, initialized, and accessed. We will learn about different operations that can be performed on strings as well as about array of strings.

4.1 INTRODUCTION

Nowadays, computers are widely used for word processing applications such as creating, inserting, updating, and modifying textual data. Besides this, we need to search for a particular pattern within a text, delete it, or replace it with another pattern. So, there is a lot that we as users do to manipulate the 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. Figure 4.1 shows the difference between character storage and string storage.

If we had declared `str` as

```
char str[5] = "HELLO";
```

then the `null` character will not be appended automatically to the character array. This is because `str` can hold only 5 characters and the characters in `HELLO` have already filled the space allocated to it.

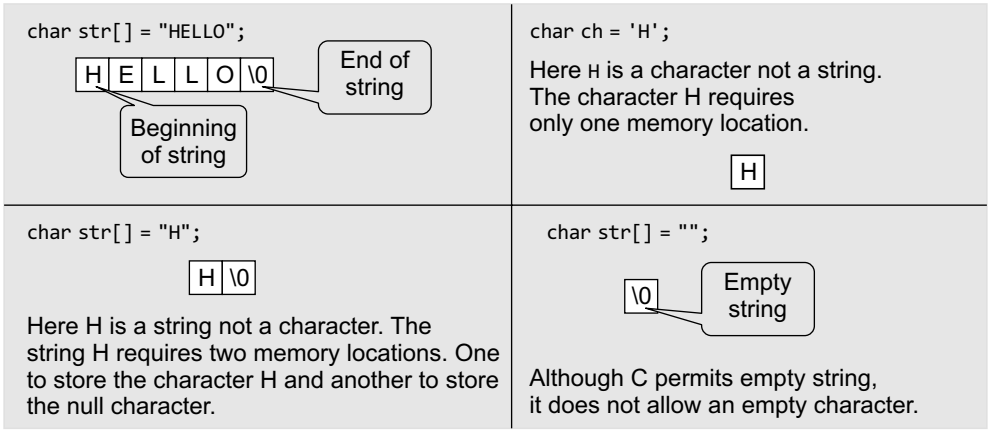


Figure 4.1 Difference between character storage and string storage

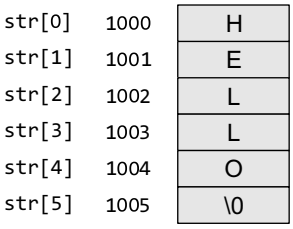


Figure 4.2 Memory representation of a character array

Programming Tip

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

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. In Fig. 4.2, 1000, 1001, 1002, etc., are the memory addresses of individual 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.

The statement

```
char str[] = "HELLO";
```

declares a constant string, as we have assigned a value to it while declaring the string. However, 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 mesg[100];` can store a maximum of 99 characters.

Till now, we have only seen one way of initializing strings. 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. Also observe that we have not mentioned the size of the string. Here, the compiler will automatically calculate the size based on the number of characters. So, in this example six memory locations will be reserved to store the string variable, `str`.

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.

4.1.1 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(),getch()or getche() function repeatedly.

Strings can be read using scanf() by writing

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

Although the syntax of using 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. This is because the moment a blank space is encountered, the string is terminated by the scanf() function. You may also specify a field width to indicate the maximum number of characters that can be read. Remember that extra characters are left unconsumed in the input buffer.

Programming Tip

Using & operand with a string variable in the scanf statement generates an error.

Unlike int, float, and char values, %s format does not require the ampersand before the variable str.

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.

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.

4.1.2 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);
```

We use the format specifier `%s` to output a string. Observe carefully that there is no '&' character used with the string variable. We may also use width and precision specifications along with `%s`. The width specifies the minimum output field width. If the string is short, the extra space is either left padded or right padded. A negative width left pads short string rather than the default right justification. The precision specifies the maximum number of characters to be displayed, after which the string is truncated. For example,

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

The above statement would print only the first three characters in a total field of five characters. Also these characters would be right justified in the allocated width. To make the string left justified, we must use a minus sign. For example,

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

Note When the field width is less than the length of the string, the entire string will be printed. If the number of characters to be printed is specified as zero, then nothing is printed on the screen.

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

```
puts(str);
```

`puts()` is a simple function that overcomes the drawbacks of the `printf()` function.

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++;
}
```

4.2 OPERATIONS ON STRINGS

In this section, we will learn about different operations that can be performed on strings.

```
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

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.

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`.

PROGRAMMING EXAMPLE

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
```

```
Step 1: [INITIALIZE] SET I=0
Step 2: Repeat Step 3 while STR[I] != NULL
Step 3:   IF STR[I] >= 'a' AND STR[I] <= '2'
            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

Converting Characters of a String into Upper/ Lower Case

We have already discussed that in the memory ASCII codes are stored instead of the real values. 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. Figure 4.4 shows an algorithm that converts the lower case characters of a string into upper case.

Note The library functions `toupper()` and `tolower()` which are defined in `ctype.h` convert a character into upper and lower case, respectively.

In the algorithm, we initialize `I` to zero. 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).

PROGRAMMING EXAMPLE

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
```

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 the destination string `s2 = s2 + s1`. Figure 4.5 shows an algorithm that appends two strings.

Note The library function `strcat(s1,s2)` which is defined in `string.h` concatenates string `s2` to `s1`.

```
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

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.

PROGRAMMING EXAMPLE

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?
```

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. Figure 4.6 shows an algorithm that compares two strings.

Note The library function `strcmp(s1, s2)` which is defined in `string.h` compares string `s1` with `s2`.

In this algorithm, we first check whether the two strings are of the same length. If not, then there is no point in moving ahead, as it straight away means that the two strings are not the same. However, 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 and print the corresponding message.

```
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

Programming Example

4. Write a program to compare two strings.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main()
{
        char str1[50], str2[50];
        int i=0, len1=0, len2=0, same=0;
        clrscr();
        printf("\n Enter the first string : ");
        gets(str1);
        printf("\n Enter the second string : ");
        gets(str2);
        len1 = strlen(str1);
        len2 = strlen(str2);
        if(len1 == len2)
        {
                while(i<len1)
                {
                        if(str1[i] == str2[i])
                                i++;
                        else break;
                }
                if(i==len1)
                {
                        same=1;
                        printf("\n The two strings are equal");
                }
        }
```

```
        if(len1!=len2)
                printf("\n The two strings are not equal");
        if(same == 0)
        {
                if(str1[i]>str2[i])
                        printf("\n String 1 is greater than string 2");
                else if(str1[i]<str2[i])
                        printf("\n String 2 is greater than string 1");
        }
        getch();
        return 0;
}
```

Output

```
Enter the first string : Hello
Enter the second string : Hello
The two strings are equal
```

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. Figure 4.7 shows an algorithm that reverses a string.

Note The library function `strrev(s1)` which is defined in `string.h` reverses all the characters in the string except the `null` character.

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 4, we swap the `i`th character of `STR` with its `j`th 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

Programming Example

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
```

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
```

```
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

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.

Programming Example

6. Write a program to extract a substring from the middle of a given string.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        char str[100], substr[100];
        int i=0, j=0, n, m;
        clrscr();
        printf("\n Enter the main string : ");
        gets(str);
        printf("\n Enter the position from which to start the substring: ");
        scanf("%d", &m);
        printf("\n Enter the length of the substring: ");
        scanf("%d", &n);
        i=m;
        while(str[i] != '\0' && n>0)
```

```
            {
                    substr[j] = str[i];
                    i++;
                    j++;
                    n--;
            }
            substr[j] = '\0';
            printf("\n The substring is : ");
            puts(substr);
            getch();
            return 0;
    }
```

Output

```
Enter the main string : Hi there
Enter the position from which to start the substring: 1
Enter the length of the substring: 4
The substring is : i th
```

```
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

Inserting a String in the Main String

The insertion operation inserts a string `S` in the main text `T` at the `K`th position. The general syntax of this operation is `INSERT(text,position,string)`. For example, `INSERT("XYZXYZ",3, "AAA") = "XYZAAAXYZ"`

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.

PROGRAMMING EXAMPLE

7. Write a program to insert a string in the main text.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        char text[100], str[20], ins_text[100];
        int i=0, j=0, k=0,pos;
        clrscr();
        printf("\n Enter the main text : ");
        gets(text);
        printf("\n Enter the string to be inserted : ");
        gets(str);
        printf("\n Enter the position at which the string has to be inserted:");
        scanf("%d", &pos);
        while(text[i]!='\0')
```

```
        {
                if(i==pos)
                {
                        while(str[k]!='\0')
                        {
                                ins_text[j]=str[k];
                                j++;
                                k++;
                        }
                }
                else
                {
                        ins_text[j]=text[i];
                        j++;
                }
                i++;
        }
        ins_text[j]='\0';
        printf("\n The new string is : ");
        puts(ins_text);
        getch();
        return0;
}
```

Output

```
Enter the main text : newsman
Enter the string to be inserted : paper
Enter the position at which the string has to be inserted: 4
The new string is: newspaperman
```

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. Figure 4.10 shows an 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
```

Figure 4.10 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. For example, if

```
TEXT = WELCOME TO THE WORLD
STRING = COME
```

In the first pass of the inner loop, we will compare COME with WELC character by character. As W and C do not match, the control will move to Step 6 and then ELCO will be compared with COME. In the fourth pass, COME will be compared with COME.

We will be using the programming code of pattern matching operation in the operations that follow.

Deleting a Substring from the Main String

The deletion operation deletes a substring from a given text. We can write it as DELETE(text, position, length). For example,

```
DELETE("ABCDXXXABCD", 4, 3) = "ABCDABCD"
```

```
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

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.

PROGRAMMING EXAMPLE

8. Write a program to delete a substring from a text.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        char text[200], str[20], new_text[200];
        int i=0, j=0, found=0, k, n=0, copy_loop=0;
        clrscr();
        printf("\n Enter the main text : ");
        gets(text);
        printf("\n Enter the string to be deleted : ");
        gets(str);
        while(text[i]!='\0')
        {
                j=0, found=0, k=i;
                while(text[k]==str[j] && str[j]!='\0')
                {
                        k++;
                        j++;
                }
```

```
                    if(str[j]=='\0')
                            copy_loop=k;
                    new_text[n] = text[copy_loop];
                    i++;
                    copy_loop++;
                    n++;
            }
            new_str[n]='\0';
            printf("\n The new string is : ");
            puts(new_str);
            getch();
            return 0;
    }
```

Output

```
Enter the main text : Hello, how are you?
Enter the string to be deleted : , how are you?
The new string is : Hello
```

Replacing a Pattern with Another Pattern in a String

The replacement operation is used to replace the pattern P_1 by another pattern P_2. This is done by writing REPLACE(text, $pattern_1$, $pattern_2$). For example,

```
("AAABBBCCC", "BBB", "X") = AAAXCCC
("AAABBBCCC", "X", "YYY")= AAABBBCC
```

```
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

In the second example, there is no change as x does not appear in the text. Figure 4.12 shows an algorithm to replace a pattern P_1 with another pattern P_2 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.

Programming Example

9. Write a program to replace a pattern with another pattern in the text.

```
#include <stdio.h>
#include <conio.h>
main()
{
        char str[200], pat[20], new_str[200], rep_pat[100];
        int i=0, j=0, k, n=0, copy_loop=0, rep_index=0;
        clrscr();
        printf("\n Enter the string : ");
        gets(str);
        printf("\n Enter the pattern to be replaced: ");
        gets(pat);
        printf("\n Enter the replacing pattern: ");
        gets(rep_pat);
        while(str[i]!='\0')
        {
                j=0,k=i;
                while(str[k]==pat[j] && pat[j]!='\0')
```

```
                    {
                            k++;
                            j++;
                    }
                    if(pat[j]=='\0')
                    {
                            copy_loop=k;
                            while(rep_pat[rep_index] !='\0')
                            {
                            new_str[n] = rep_pat[rep_index];
                            rep_index++;
                            n++;
                            }
                    }
                    new_str[n] = str[copy_loop];
                    i++;
                    copy_loop++;
                    n++;
            }
            new_str[n]='\0';
            printf("\n The new string is : ");
            puts(new_str);
            getch();
            return 0;
    }
```

Output

```
Enter the string : How ARE you?
Enter the pattern to be replaced : ARE
Enter the replacing pattern : are
The new string is : How are you?
```

4.3 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. 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 will specify the length of every individual string. So here, we will allocate space for 20 names where each name can be a maximum 30 characters long.

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.

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

```
Step 1: [INITIALIZE] SET I=0
Step 2: Repeat Step 3 while I< N
Step 3:     Apply Process to NAMES[I]
        [END OF LOOP]
Step 4: EXIT
```

Figure 4.14 Algorithm to process individual string from an array of strings

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. Figure 4.14 shows an algorithm to process individual string from an array of strings.

In Step 1, we initialize the index variable `I` to zero. In Step 2, a `while` loop is executed until all the strings in the array are accessed. In Step 3, each individual string is processed.

Programming Examples

10. Write a program to sort the names of students.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main()
{
        char names[5][10], temp[10];
        int i, n, j;
        clrscr();
        printf("\n Enter the number of students : ");
        scanf("%d", &n);
        for(i=0;i<n;i++)
        {
                printf("\n Enter the name of student %d : ", i+1);
                gets(names[i]);
        }
        for(i=0;i<n;i++)
        {
                for(j=0;j<n-i-1;j++)
                {
                        if(strcmp(names[j], names[j+1])>0)
                        {
                                strcpy(temp, names[j]);
                                strcpy(names[j], names[j+1]);
                                strcpy(names[j+1], temp);
                        }
                }
        }
        printf("\n Names of the students in alphabetical order are : ");
        for(i=0;i<n;i++)
                puts(names[i]);
        getch();
        return 0;
}
```

Output

```
Enter the number of students : 3
Enter the name of student 1 : Goransh
Enter the name of student 2 : Aditya
Enter the name of student 3 : Sarthak
Names of the students in alphabetical order are : Aditya Goransh Sarthak
```

11. Write a program to read multiple lines of text and then count the number of characters, words, and lines in the text.

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

```
#include <conio.h>
int main()
{
        char str[1000];
        int i=0, word_count = 1, line_count =1, char_count = 1;
        clrscr();
        printf("\n Enter a '*' to end");
        printf("\n **************");
        printf("\n Enter the text : ");
        scanf("%c", &str[i]);
        while(str[i] != '*')
        {
                i++;
                scanf("%c", &str[i]);
        }
        str[i] = '\0';
        i=0;
        while(str[i] != '\0')
        {
                if(str[i] == '\n' || i==79)
                        line_count++;
                if(str[i] == ' ' &&str[i+1] != ' ')
                        word_count++;
                char_count++;
                i++;
        }
        printf("\n The total count of words is : %d", word_count);
        printf("\n The total count of lines is : %d", line_count);
        printf("\n The total count of characters is : %d", char_count);
        return 0;
}
```

Output

```
Enter a '*' to end
**************
Enter the text : Hi there*
The total count of words is : 2
The total count of lines is : 1
The total count of characters is : 9
```

12. Write a program to find whether a string is a palindrome or not.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        char str[100];
        int i = 0, j, length = 0;
        clrscr();
        printf("\n Enter the string : ");
        gets(str);
        while(str[i] != '\0')
        {
                length++ ;
                i++ ;
        }
        i=0;
        j = length - 1;
        while(i <= length/2)
        {
```

```
                    if(str[i] == str[j])
                    {
                            i++;
                            j--;
                    }
                    else
                            break;
            }
            if(i>=j)
                    printf("\n PALINDROME");
            else
                    printf("\n NOT A PALINDROME");
            return 0;
    }
```

Output

```
Enter the string: madam
PALINDROME
```

4.4 POINTERS AND STRINGS

In C, strings are treated as arrays of characters that are terminated with a binary zero character (written as `'\0'`). Consider, for example,

```
char str[10];
str[0] = 'H';
str[1] = 'i';
str[2] = '!':
str[3] = '\0';
```

C provides two alternate ways of declaring and initializing a string. First, you may write

```
char str[10] = {'H', 'i', '!', '\0'};
```

But this also takes more typing than is convenient. So, C permits

```
char str[10] = "Hi!";
```

When the double quotes are used, a `null` character (`'\0'`) is automatically appended to the end of the string.

When a string is declared like this, the compiler sets aside a contiguous block of the memory, i.e., 10 bytes long, to hold characters and initializes its first four characters as `Hi!\0`.

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
```

In this program, we declare a character pointer `*pstr` to show the string on the screen. We then point the pointer `pstr` to `str`. Then, we print each character of the string using the `while` loop. Instead of using the `while` loop, we could straightaway use the function `puts()`, as shown below

```
puts(pstr);
```

The function prototype for `puts()` is as follows:

```
int puts(const char *s);
```

Here the `const` modifier is used to assure that the function dose not modify the contents pointed to by the source pointer. The address of the string is passed to the function as an argument.

The parameter passed to `puts()` is a pointer which is nothing but the address to which it points to or simply an address. Thus, writing `puts(str)` means passing the address of `str[0]`. Similarly when we write `puts(pstr);` we are passing the same address, because we have written `pstr = str;`.

Consider another program that reads a string and then scans each character to count the number of upper and lower case characters entered.

```
#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
```

Programming Examples

13. Write a program to copy a string into another string.

```
#include <stdio.h>
int main()
{
        char str[100], copy_str[100];
        char *pstr, *pcopy_str;
        pstr = str;
        pcopy_str = copy_str;
        printf("\n Enter the string : ");
        gets(str);
        while(*pstr != '\0')
        {
```

```
                *pcopy_str = *pstr;
                pstr++, pcopy_str++;
        }
        *pcopy_str = '\0';
        printf("\n The copied text is : ");
        while(*pcopy_str != '\0')
        {
                printf("%c", *pcopy_str);
                pcopy_str++;
        }
        return 0;
}
```

Output

```
Enter the string : C Programming
The copied text is : C Programming
```

14. Write a program to concatenate two strings.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        char str1[100], str2[100], copy_str[200];
        char *pstr1, *pstr2, *pcopy_str;
        clrscr();
        pstr1 = str1;
        pstr2 = str2;
        pcopy_str = copy_str;
        printf("\n Enter the first string : ");
        gets(str1);
        printf("\n Enter the second string : ");
        gets(str2);
        while(*pstr1 != '\0')
        {
                *pcopy_str = *pstr1;
                pcopy_str++, pstr1++;
        }
        while(*pstr2 != '\0')
        {
                *pcopy_str = *pstr2;
                pcopy_str++, pstr2++;
        }
        *pcopy_str = '\0';
        printf("\n The concatenated text is : ");
        while(*pcopy_str != '\0')
        {
                printf("%c", *pcopy_str);
                pcopy_str++;
        }
        return 0;
}
```

Output

```
Enter the first string : Data Structures Using C by
Enter the second string : Reema Thareja
The concatenated text is : Data Structures Using C by Reema Thareja
```

Points to Remember

- A string is a null-terminated character array.
- Individual characters of strings can be accessed using a subscript that starts from zero.
- All the characters of a string are stored in successive memory locations.
- Strings can be read by a user using three ways: using `scanf()` function, using `gets()` function, or using `getchar()` function repeatedly.
- The `scanf()` function terminates as soon as it finds a blank space.
- 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.
- Strings can also be read by calling `getchar()` repeatedly to read a sequence of single characters.
- Strings can be displayed on the screen using three ways: using `printf` function, using `puts()` function, or using `putchar()` function repeatedly.
- C standard library supports a number of pre-defined functions for manipulating strings or changing the contents of strings. Many of these functions are defined in the header file `string.h`.
- Alternatively we can also develop functions which perform the same task as the pre-defined string handling functions. The most basic function is the `length` function which returns the number of characters in a string.
- Name of a string acts as a pointer to the string. In the declaration `char str[5] = "hello";` `str` is a pointer which holds the address of the first character, i.e., `'h'`.
- An array of strings can be declared as `char strings [20][30];` where the first subscript denotes the number of strings and the second subscript denotes the length of every individual string.

Exercises

Review Questions

1. What are strings? Discuss some of the operations that can be performed on strings.
2. Explain how strings are represented in the main memory.
3. How are strings read from the standard input device? Explain the different functions used to perform the string input operation.
4. Explain how strings can be displayed on the screen.
5. Explain the syntax of `printf()` and `scanf()`.
6. List all the substrings that can be formed from the string 'ABCD'.
7. What do you understand by pattern matching? Give an algorithm for it.
8. Write a short note on array of strings.
9. Explain with an example how an array of strings is stored in the main memory.
10. Explain how pointers and strings are related to each other with the help of a suitable program.
11. If the substring function is given as `SUBSTRING (string, position, length)`, then find `S(5,9)` if `S = "Welcome to World of C Programming"`
12. If the index function is given as `INDEX(text, pattern)`, then find `index(T, P)` where `T = "Welcome to World of C Programming"` and `P = "of"`
13. Differentiate between `gets()` and `scanf()`.
14. Give the drawbacks of getchar() and scanf().
15. Which function can be used to overcome the shortcomings of `getchar()` and `scanf()`?
16. How can `putchar()` be used to print a string?
17. Differentiate between a character and a string.
18. Differentiate between a character array and a string.

Programming Exercises

1. Write a program in which a string is passed as an argument to a function.
2. Write a program in C to concatenate first `n` characters of a string with another string.
3. Write a program in C that compares first `n` characters of one string with first `n` characters of another string.
4. Write a program in C that removes leading and trailing spaces from a string.
5. Write a program in C that replaces a given character with another character in a string.

6. Write a program to count the number of digits, upper case characters, lower case characters, and special characters in a given string.
7. Write a program to count the total number of occurrences of a given character in the string.
8. Write a program to accept a text. Count and display the number of times the word 'the' appears in the text.
9. Write a program to count the total number of occurrences of a word in the text.
10. Write a program to find the last instance of occurrence of a sub-string within a string.
11. Write a program to input an array of strings. Then, reverse the string in the format shown below.
    `"HAPPY BIRTHDAY TO YOU"` should be displayed as `"YOU TO BIRTHDAY HAPPY"`
12. Write a program to append a given string in the following format.
    `"GOOD MORNING MORNING GOOD"`
13. Write a program to input a text of at least two paragraphs. Interchange the first and second paragraphs and then re-display the text on the screen.
14. Write a program to input a text of at least two paragraphs. Construct an array `PAR` such that `PAR[I]` contains the location of the ith paragraph in text.
15. Write a program to convert the given string `"GOOD MORNING"` to `"good morning"`.
16. Write a program to concatenate two given strings `"Good Morning"` and `"World"`. Display the resultant string.
17. Write a program to check whether the two given strings `"Good Morning"` and `"GOOD MORNING"` are same.
18. Write a program to convert the given string `"hello world"` to `"dlrow olleh"`.
19. Write a program to extract the string `"od Mo"` from the given string `"Good Morning"`.
20. Write a program to insert `"University"` in the given string `"Oxford Press"` so that the string should read as `"Oxford University Press"`.
21. Write a program to read a text, delete all the semi-colons it has, and finally replace all `'.'` with a `','`.
22. Write a program to copy the last `n` characters of a character array in another character array. Also, convert the lower case letters into upper case letters while copying.
23. Write a program to rewrite the string `"Good Morning"` to `"Good Evening"`.
24. Write a program to read and display names of employees in a department.
25. Write a program to read a line until a newline is entered.
26. Write a program to read a short story. Rewrite the story by printing the line number before the starting of each line.
27. Write a program to enter a text that contains multiple lines. Display the `n` lines of text starting from the `m`th line.
28. Write a program to check whether a pattern exists in a text. If it does, delete the pattern and display it.
29. Write a program to insert a new name in the string array `STUD[][]`, assuming that names are sorted alphabetically.
30. Write a program to delete a name in the string array `STUD[][]`, assuming that names are sorted alphabetically.

Multiple-choice Questions

1. `Insert("XXXYYYZZZ", 1, "PPP") =`
   (a) PPPXXXYYYZZZ
   (b) XPPPXXYYYZZZ
   (c) XXXYYYZZZPPP
2. `Delete("XXXYYYZZZ", 4,3) =`
   (a) XXYZ (b) XXXYYZZ
   (c) XXXYZZ
3. If `str[] = "Welcome to the world of programming"`, then `SUBSTRING(str, 15, 5) =`
   (a) world (b) programming
   (c) welcome (d) none of these
4. `strcat()` is defined in which header file?
   (a) `ctype.h` (b) `stdio.h`
   (c) `string.h` (d) `math.h`
5. A string can be read using which function(s)?
   (a) `gets()` (b) `scanf()`
   (c) `getchar()` (d) all of these
6. `Replace("XXXYYYZZZ", "XY", "AB") =`
   (a) XXABYYZZZ (b) XABYYYZZZ
   (c) ABXXXYYYZZ
7. The index of U in Oxford University Press is?
   (a) 5 (b) 6
   (c) 7 (d) 8

8. s1 = "HI", s2 = "HELLO", s3 = "BYE". How can we concatenate the three strings?
   (a) strcat(s1,s2,s3)
   (b) strcat(s1(strcat(s2,s3)))
   (c) strcpy(s1, strcat(s2,s3))
9. strlen("Oxford University Press") is ?
   (a) 22 (b) 23
   (c) 24 (d) 25
10. Which function adds a string to the end of another string?
   (a) stradd() (b) strcat()
   (c) strtok() (d) strcpy()

True or False

1. String `Hello World` can be read using `scanf()`.
2. A string when read using `scanf()` needs an ampersand character.
3. The `gets()` function takes the starting address of a string which will hold the input.
4. `tolower()` is defined in `ctype.h` header file.
5. If S_1 and S_2 are two strings, then the concatenation operation produces a string which contains the characters of S_2 followed by the characters of S_1.
6. Appending one string to another string involves copying the contents of the source string at the end of the destination string.
7. `S1<S2`, when in dictionary order, `S1` precedes `S2`.
8. If `S1 = "GOOD MORNING"`, then `Substr_Right (S1, 5) = MORNING`.
9. `Replace ("AAABBBCCC", "X", "YYY")= AAABBBCC`.
10. Initializing a string as `char str[] = "HELLO"`; is incorrect as a null character has not been explicitly added.
11. The `scanf()` function automatically appends a null character at the end of the string read from the keyboard.
12. String variables can be present either on the left or on the right side of the assignment operator.
13. When a string is initialized during its declaration, the string must be explicitly terminated with a null character.
14. `strcmp("and", "ant");` will return a positive value.
15. Assignment operator can be used to copy the contents of one string into another.

Fill in the blanks

1. Strings are _____.
2. Every string is terminated with a _____.
3. If a string is given as `"AB CD"`, the length of this string is _____.
4. The subscript of a string starts with _____.
5. Characters of a string are stored in _____ memory locations.
6. `char mesg[100];` can store a maximum of _____ characters.
7. _____ function terminates as soon as it finds a blank space.
8. The ASCII code for `A-Z` varies from _____.
9. `toupper()` is used to _____.
10. `S1>S2` means _____.
11. The function to reverse a string is _____.
12. If `S1 = "GOOD MORNING"`, then `Substr_Left (S1, 7) =` _____.
13. `INDEX("Welcome to the world of programming", "world") =` _____.
14. _____ returns the position in the string where the string pattern first occurs.
15. strcmp(str1, str2) returns 1 if _____.
16. _____ function computes the length of a string.
17. Besides printf(), _____ function can be used to print a line of text on the screen.