String Handling and I/O Functions

Key Concepts

- **String handling using arrays**
- **Memory allocation for strings**
- **Input/output operations on strings**
- **Console functions for Character I/O**
    - getchar()
    - putchar()
- **Stream functions for I/O operations**
    - Input functions - get(), getline()
    - Output functions - put(), write()

We have learnt that array is the effective mechanism to handle large number of homogeneous type of data. Most of the programs that we discussed are used to process numeric type data. We know that there are string type data also. In this chapter we will see how string data are stored and processed. Also we will discuss some built-in functions to perform input/output operations on string and character data.

9.1 String handling using arrays

We know that string is a kind of literal in C++ language. It appears in programs as a sequence of characters within a pair of double quotes. Imagine that you are asked to write a program to store your name and display it. You have learned that variables are required to store data. Let us take the identifier `my_name` as the variable. Remember that in C++, a variable is to be declared before it is used. A declaration statement is required for this and it begins with a data type. Which data type should be used to declare a variable to hold string data? There is no basic data type to represent string data. You may think of **`char`** data type. But note that the variable declared using `char` can hold only one character. Here we have to input string which is a sequence of characters.

Let us consider a name "Niketh". It is a string consisting of six characters. So it cannot be stored in a variable of `char` type. But we know that an array of `char` type can hold more than one character. So, we declare an array as follows:

```
    char  my_name[10];
```

It is sure that ten contiguous locations, each with one byte size, will be allocated for the array named `my_name`. If we follow the usual array initialization method, we can store the characters in the string "Niketh" as follows:

```
    char my_name[10]={'N', 'i', 'k', 'e', 't', 'h'};
```

Figure 9.1 shows the memory allocation for the above declared character array. Note that we store the letters in the string, separated by commas. If we want to input the same data, the following C++ statement can be used:

```
     for (int i=0; i<6; i++)
         cin >> my_name[i];
```

During the execution of this statement, we have to input six letters of "Niketh" one after the other separated by **Space bar**, **Tab** or **Enter** key. The memory allocation in both of these cases may be shown as follows:

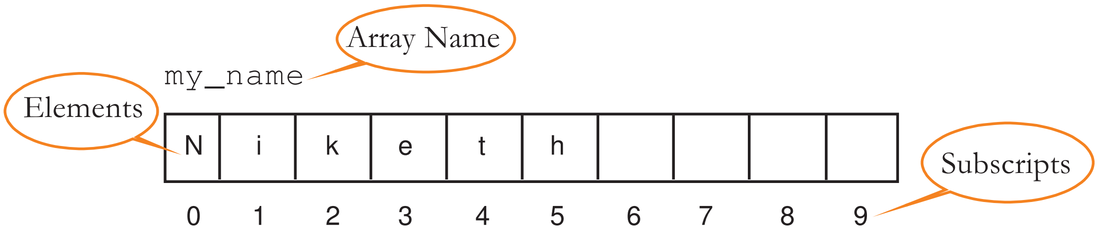

Fig. 9.1 : Memory allocation for the character array

So, let us conclude that a character array can be used to store a string, since it is a sequence of characters. However, it is true that we do not get the feel of inputting a string. Instead, we input the characters constituting the string one by one.

In C++, character arrays have some privileges over other arrays. Once we declare a character array, the array name can be considered as an ordinary variable that can hold string data. Let's say that a character array name is equivalent to a string variable. Thus your name can be stored in the variable `my_name` (the array name) using the following statement:

```
    cin >> my_name;
```

It is important to note that this kind of usage is wrong in the case of arrays of other data types. Now let us complete the program. It will be like the one given in Program 9.1.

Program 9.1: To input a string and display

```
#include <iostream>
using namespace std;
int main()
{
     char my_name[10];
     cout << "Enter your name: ";
     cin >> my_name;
     cout << "Hello " << my_name;
}
```

On executing this program you will get the output as shown below.

```
    Enter your name: Niketh
    Hello Niketh
```

Note that the string constant is not `"Hello"`, but `"Hello "` (a white space is given after the letter **o**).

Let us do

Run Program 9.1 and input your full name by expanding the initials if any, and check whether the output is correct or not. If your name contains more than 10 characters, increase the size of the array as needed.

9.2 Memory allocation for strings

We have seen how memory is allocated for an array of characters. As Figure 9.1 shows, the memory required depends upon the number of characters stored. But if we input a string in a character array, the scene will be different. If we run Program 9.1 and input the string `Niketh`, the memory allocation will be as shown in Figure 9.2.

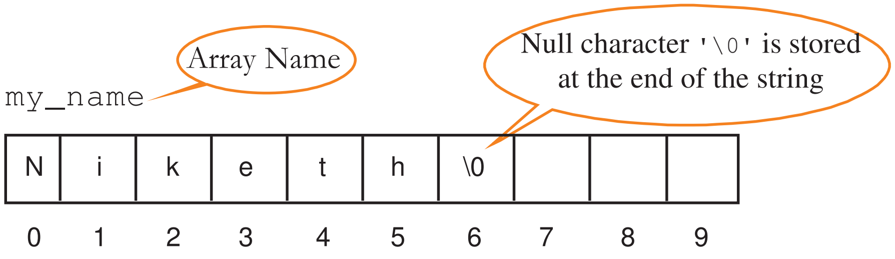

Fig. 9.2 : Memory allocation for the character array

Note that a null character `'\0'` is stored at the end of the string. This character is used as the string terminator and added at the end automatically. Thus we can say that memory required to store a string will be equal to the number of characters in the string plus one byte for null character. In the above case, the memory used to store the string `Niketh` is seven bytes, but the number of characters in the string is only six.

As in the case of variable initialization, we can initialize a character array with a string as follows:

```
char my_name[10] = "Niketh";
char str[] = "Hello World";
```

In the first statement 10 memory locations will be allocated and the string will be stored with null character as the delimiter. The last three bytes will be left unused. But for the second statement, size of the array is not specified and hence only 12 bytes will be allocated (11 bytes for the string and 1 for `'\0'`).

9.3 Input/Output operations on strings

Program 9.1 contains input and output statements for string data. Let us modify the declaration statement by changing the size of the array to 20. If we run the program by entering the name `Maya Mohan`, the output will be as follows:

```
Enter your name: Maya Mohan
Hello Maya
```

Note that though there is enough size for the array, we get only the word `"Maya"` as the output. Why does this happen?

Let us have a close look at the input statement: `cin>>my_name;`. We have experienced that only one data item can be input using this statement. A white space is treated as a separator of data. Thus, the input `Maya Mohan` is treated as two data items, `Maya` and `Mohan` separated by white space. Since there is only one input operator (**>>**) followed by a variable, the first data (i.e., `Maya`) is stored. The white space after `"Maya"` is treated as the delimiter.

So, the problem is that we are unable to input strings containing white spaces. C++ language gives a solution to this problem by a function, named **`gets()`**. The function `gets()` is a console input function used to accept a string of characters including white spaces from the standard input device (keyboard) and store it in a character array.

The string variable (character array name) should be provided to this function as shown below:

```
gets(character_array_name);
```

When we use this function, we have to include the library file `stdio.h` in the program. Let us modify Program 9.1, by including the statement `#include<cstdio>`, and replacing the statement `cin>>my_name;` by `gets(my_name);`. After executing the modified program, the output will be as follows:

```
Enter your name: Maya Mohan
Hello Maya Mohan
```

The output shows the entire string that we input. See the difference between `gets()` and `cin`.

Though we are not using the concept of subscripted variable for the input and output of strings, any element in the array can be accessed by specifying its subscript along with the array name. We can access the first character of the string by `my_name[0]`, fifth character by `my_name[4]` and so on. We can even access the null character (`'\0'`) by its subscript. The following program illustrates this idea.

Program 9.2: To input a string and count the vowels in a string

```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
 {
  char str[20];
  int vow=0;
  cout<<"Enter a string: ";
  gets(str);
  for(int i=0; str[i]!='\0'; i++)
     switch(str[i])
     {
           case 'a':
           case 'e':
           case 'i':
           case 'o':
           case 'u': vow++;
     }
  cout<<"No. of vowels in the string "<<str<<" is "<<vow;
  return 0;
 }
```

Header file for using gets( ) function

Condition will be true as long as the null character is not retrieved

Each character in the array will be compared with the constants for matching

If we run Program 9.2 by inputting the string “hello guys”, the following output can be seen:

```
Enter a string: hello guys
No. of vowels in the string hello guys is 3
```

Now, let us analyse the program and see how it works to give this output.

- In the beginning, the `gets()` function is used and so we can input the string `"hello guys"`.
- The body of the `for` loop will be executed as long as the element in the array, referenced by the subscript `i`, is not the null character (`'\0'`). That is, the body of the loop will be executed till the null character is referenced.

- The body of the loop contains only a `switch` statement. Note that, no statements are given against the first four cases of the `switch`. In the last case, the variable `vow` is incremented by 1. You may think that this is required for all the cases. Yes, you are right. But you should use the `break` statement for each case to exit the `switch` after a match. In this program the action for all the cases are the same and that is why we use this style of code.
- While the `for` loop iterates, the characters will be retrieved one by one for matching against the constants attached to the cases. Whenever a match is found, the variable `vow` is incremented by 1.
- As per the input string, matches occur when the value of `i` becomes 1, 4 and 7. Thus, the variable `vow` is incremented by 1 three times and we get the correct output.

We have seen how `gets()` function facilitates input of strings. Just like the other side of a coin, C++ gives a console function named **`puts()`** to output string data. The function `puts()` is a console output function used to display a string data on the standard output device (monitor). Its syntax is:

```
puts(string_data);
```

The string constant or variable (character array name) to be displayed should be provided to this function. Observe the following C++ code fragment:

```
char str[10] = "friends";
puts("hello");
puts(str);
```

The output of the above code will be as follows:

```
hello
friends
```

Note that the string `"friends"` in the character array `str[10]` is displayed in a new line. Try this code using `cout<<“hello”;` and `cout<<str;` instead of the `puts()` functions and see the difference. The output will be in the same line without a space in between them in the case of `cout` statement.

Let us do

Predict the output, if the input is “HELLO GUYS” in Program 9.2. Execute the program with this input and check whether you get correct output. Find out the reason for difference in output. Modify the program to get the correct output for any given string.

9.4 Console functions for character I/O

We have discussed functions for input/output operations on strings. C++ also provides some functions for performing input/ouput operations on characters. These functions require the inclusion of header file **`cstdio`** (`stdio.h` in Turbo C++ IDE) in the program.

getchar()

This function returns the character that is input through the keyboard. The character can be stored in a variable as shown in the example given below:

```
    char ch = getchar();
```

We have seen `puts()` function and its advantage in string output. Let us have a look at a function used to output character data.

putchar()

This function displays the character given as the argument on the standard output unit (monitor). The argument may be a character constant or a variable. If an integer value is given as the argument, it will be considered as an ASCII value and the corresponding character will be displayed. The following code segment illustrates the use of `putchar()` function.

```
    char ch = 'B';   //assigns 'B' to the variable ch
    putchar(ch);     //displays 'B' on the screen
    putchar('c');    //displays 'c' on the screen
    putchar(97);     //displays 'a' on the screen
```

Program 9.3 illustrates the working of these functions. This program allows inputting a string and a character to be searched. It displays the number of occurrences of a character in the string.

Program 9.3: To search for a character in a string using console functions

```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
   char str[20], ch;
   int i, num=0;
   puts("Enter a string:"); //To print '\n' after the string
   gets(str);//To accept a string with white spaces
```

```
    cout<<"Enter the character to be searched: ";
    ch=getchar(); //To input the character to be searched
    /* A loop to search for the character and count its
       occurrences in the string. Search will be
       terminated when a null character is found   */
    for(i=0; str[i]!='\0'; i++)
       if (str[i]==ch)
          num++;
    cout<<"The string \'"<<str<<"\' uses the character \'";
    putchar(ch);
    cout<<"\' ")<<num<<" times";
    return 0;
}
```

Program 9.3 uses all the console functions we have discussed. The following is a sample output of this program:

```
Enter a string:
I have a Dream
Enter the character to be searched: a
The string 'I have a Dream' uses the character 'a' 3 times
```

Check yourself

1. Which character is used to delimit the string in memory?
2. Write the statement to declare a variable for storing "Save Earth".
3. Name the header file required for using console I/O functions.
4. How many bytes are required to store the string "Be Positive"?
5. How does `puts("hello");` differ from `cout<<"hello";`?

9.5 Stream functions for I/O operations

C++ provides another facility to perform input/output operations on character and strings. It is in the form of functions that are available in the header file **`iostream`**. These functions are generally called stream functions since they allow a stream of bytes (data) to flow between memory and objects. Devices like the keyboard and the monitor are referenced as objects in C++. Let us discuss some of these functions.

A. Input functions

These functions allow the input of character and string data. The input functions such as **`get()`** and **`getline()`** allow a stream of bytes to flow from input object into the memory. The object **`cin`** is used to refer to keyboard and hence whenever we input data using keyboard, these functions are called or invoked using this object as **`cin.get()`** and **`cin.getline()`**. Note that a period symbol (.), called ***dot operator*** is used between the object `cin` and the function.

i. get()

It can accept a single character or multiple characters (string) through the keyboard. To accept a string, an array name and size are to be given as arguments. Following code segment illustrates the usage of this function.

```
char ch, str[10];
ch=cin.get(ch); //accepts a character and stores in ch
cin.get(ch);    //equivalent to the above statement
cin.get(str,10); //accepts a string of maximum 10 characters
```

ii. getline()

It accepts a string through the keyboard. The delimiter will be Enter key, the number of characters or a specified character. This function uses two syntaxes as shown in the code segment given below.

```
char ch, str[10];
int len;
cin.getline(str,len);        // With 2 arguments
cin.getline(str,len,ch);     // With 3 arguments
```

In the first usage, `getline()` function has two arguments - a character array (here it is, `str`) and an integer (`len`) that represents maximum number of characters that can be stored. In the second usage, a delimiting character (content of `ch`) can also be given along with the number of characters. While inputting the string, only (`len-1`) characters, or characters upto the specified delimiting character, whichever occurs first will be stored in the array.

B. Output functions

Output functions like **`put()`** and **`write()`** allow a stream of bytes to flow from memory into an output object. The object **`cout`** is used with these functions since we use the monitor for the output.

i. put()

It is used to display a character constant or the content of a character variable given as argument.

```
char ch='c';
cout.put(ch);     //character 'c' is displayed
cout.put('B');    //character 'B' is printed
cout.put(65);     //character 'A' is printed
```

ii. `write()`

This function displays the string contained in the argument. For illustration see the example given below.

```
    char  str[10]="hello";
    cout.write(str,10);
```

The above code segment will dispaly the string `hello` followed by 5 white spaces, since the second argument is 10 and the number of characters in the string is 5.

Program 9.4: To illustrate the working of stream input/output functions

```
#include  <iostream>
#include <cstring> //To use strlen() function
using namespace std;
int main()
{
    char ch, str[20];
    cout<<"Enter a character: ";
    cin.get(ch); //To input a character to the variable ch
    cout<<"Enter a string: ";
    cin.getline(str,10,'.');     //To input the string
    cout<<"Entered character is:\t";
    cout.put(ch);              //To display the character
    cout.write("\nEntered string is:",20);
    cout.write(str,strlen(str));
    return 0;
}
```

On executing Program 9.4, the following output will be obtained:

```
   Enter a character: p
   Enter a string: hello world
   Entered character is:     p
   Entered string is:
   hello wo
```

Let us discuss what happens when the program is executed. In the beginning, `get()` function allows to input a character, say `p`. When the function `getline()` is executed, we can input a string, say `hello world`. The `put()` function is then executed to display the character `p`. Observe that the `write()` function displays only `hello wo` in a new line. In the `getline()` function, we specified the integer 10 as the maximum number of characters to be stored in the array `str`. Usually 9 characters will be stored, since one byte is reserved for `'\0'` character as the string terminator. But the output shows only 8 characters including white space. This is because, the Enter key followed by the character input (p) for the `get()` function, is stored as the `'\n'` character in the first location of `str`. That is why, the string, `hello wo` is displayed in a new line.

If we run the program, by giving the input `hello.world`, the output will be as follows: Observe the change in the content of `str`.

```
Enter a character: a
Enter a string: hello.world
Entered character is:     a
Entered string is:
hello
```

Note the dot character (.)

The change has occurred because the `getline()` function accepts only the characters that appear before the dot symbol.

Be careful while using these functions, because pressing of any key does matter a lot in the input operation. So you may not get the outputs as you desire. Another point you have to notice is that, **strlen()** function is used in the write() function. Instead of using this function, you can provide a number like 10 or 20. But the output will be the string you input, followed by some ASCII characters, if the number of characters is less than the given number. When you use **strlen()**, you are actually specifying the exact number of characters in the string. More about this function will be discussed in chapter 10. You can use this function only if you include **cstring** file.

Let us do

The following table compares the stream functions. But it is not complete. Fill up the table and check whether your entries are correct by comparing with that of your friends.

Comparison Aspect	*Console Functions*	*Stream Functions*
Header file required	....	....
Usage format	Mention the function name with required data or variable within parentheses	Use object name followed by dot operator and function name with required data or variable.
Device reference	Keyboard or monitor is not mentioned	....
Examples	....	....

Check yourself

1. Name the stream function to input a character data.
2. Write a C++ statement to display the string "Smoking is injurious to health" using `write()` function.
3. Name the header file required for using stream I/O functions.
4. Write down the syntax of `getline()` function.

Let us sum up

Array of characters is used to handle strings in C++ programs. While allocating memory for string, a null character (`'\0'`) is placed as the delimiter. Different console functions are available to perform input/output operations on strings. These functions are available in the header file `cstdio`. The header file `iostream` provides some stream function for the input and output of strings.

Learning outcomes

After the completion of this chapter the learner will be able to

- use character arrays for string handling.
- use various built-in functions for I/O operations on character and string data.
- compare console functions and stream functions.

Lab activity

1. Write a program to input a string and find the number of uppercase letters, lowercase letters, digits, special characters and white spaces.
2. Write a program to count the number of words in a sentence.
3. Write a program to input a string and replace all lowercase vowels by the corresponding uppercase letters.
4. Write a program to input a string and display its reversed string using console I/O functions only. For example, if the input is "AND", the output should be "DNA".

5. Write a program to input a word (say COMPUTER) and create a triangle as follows:

```
C
C   O
C   O   M
C   O   M   P
C   O   M   P   U
C   O   M   P   U   T
C   O   M   P   U   T   E
C   O   M   P   U   T   E   R
```

6. Write a program to input a line of text and display the first characters of each word. Use only console I/O functions. For example, if the input is "Save Water, Save Nature", the output should be "SWSN"
7. Write a program to check whether a string is palindrome or not. (A string is said to be plaindrome if it is the same as the string constituted by reversing the characters of the original string. eg. "MALAYALAM")

Sample questions

Very short answer type

1. What will the output of the statement: `putchar(97);` be?
2. Distinguish between console functions and stream functions.
3. Write a C++ statement to input the string “Computer” using `get()` function.
4. Write down the output of the following code segment:

```
puts(“hello”);
puts(“friends”);
```

Short answer type

1. Predict the output of the following code segment:

```
char str[] = “Program”;
for (int i=0; str[i] != ‘\0’; ++i)
    {
    putchar(str[i]);
    putchar(‘-’);
    }
```

2. Identify the errors in the following program and give reason for each.

```
#include<iostream>
using namespace std;
int main()
{
        char ch, str[10];
        write("Enter a character");
        ch=getchar();
        puts("Enter a string");
        cin.getline(str);
        cout<< "The data entered are " <<ch;
        putchar(str);
}
```

3. Observe the following functions. If the statement is valid, explain what happens when they are executed. If invalid, state reasons for it. (*Assume that the variables are valid*)

   (a) `getchar(ch);`　　(b) `gets(str[5]);`

   (c) `putchar("hello");`　　(d) `cin.getline(name, 20, ',');`

   (e) `cout.write("hello world", 10);`

4. Read the following statements:

```
        char name[20];
        cin>>name;
        cout<<name;
```

   What will be the output if you input the string "Sachin Tendulkar"? Justify your answer. Modify the code to get the entered string as the output.

Long answer type

1. Explain the console I/O functions with examples.
2. Briefly describe the stream I/O functions with examples.