Lab 09

Objectives

After performing this lab, students will be able to

- Understand string type variables and use them in programs

String data type

A string is called as a collection/sequence of characters such as “Hi” or “I am a student”. Strings are enclosed within double quotes “”. In order to use the string data type, the C++ string header **<string>** must be included at the top of the program. Also, you’ll need to include using **namespace std;** to make the short name string visible instead of requiring the cumbersome **std::string**.

There are two types of strings commonly used in C++:

1. C-strings (C-style Strings)
2. Strings that are objects of **string** class (The Standard C++ Library **string** class)

C-strings

In C programming, the collection of characters is stored in the form of arrays. This is also supported in C++ programming. Hence it's called **C-strings**.

C-strings are arrays of type **char** terminated with **null** character, that is, **\0** (ASCII value of **null** character is 0). For example,

```
char str[] = "C++";
```

In the above code snippet, **str** is a string and it holds 4 characters. Although the literal "C++" has 3 character, the **null** character **\0** is added to the end of the string automatically to indicate the end of string.

C++ string

Most of the time we will be using C++ string class (provided by the C++ library) because it is easier to use than C-strings.

Just like creating variables of other data types, to create a string we first declare it, then we can store a value in it, like shown below:

```
string testString;
testString = "This is a string.";
```

We can combine these two statements into one line:

```
string testString = "This is a string.";
```

Often, we use strings as output in the **cout** statement. So, if we write:

```
cout << testString << endl;
```

It will be the same as writing:

```
cout << "This is a string." << endl;
```

Getting input into a string variable

We cannot use the standard **cin** statement for getting input into a string variable. Consider the following code:

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
  string name;
  cout << "Enter your name: ";
  cin >> name;
  cout << "Name is " << name << endl;

  return 0;
}
```

Output:

```
Enter your name: James Gosling
Name is James
```

Here, only the first word “James” is assigned to the variable. The text after the space after the first word is ignored. To get the entire line of text, we should use a built-in function **getline()** as shown below:

```
int main()
{
  string name;
  cout << "Enter your name: ";
  getline(cin, name);
  cout << "Name is " << name << endl;

  return 0;
}
```

Output 1:

```
Enter your name: James Gosling
Name is James Gosling
```

Output 2:

```
Enter your name: John Anthony Charles
Name is John Anthony Charles
```

Strings – Basic operations (built-in functions of string class)

Counting the number of characters in a string.

The **length( )** method returns the *number of characters* in a string, including spaces and punctuation. Like many of the string operations, length is a **member (built-in)** function, and we invoke member functions using **dot** notation. The string that is the receiver is to the left of the **dot**, the member function we are invoking is to the right, (e.g. **str.length( )**). In such an expression, we are requesting the length from the string variable **str**.

```
#include <string>
#include <iostream>

using namespace std;

int main() {
    string small, large;
    small = "I am short";
    large = "I, friend, am a long and elaborate string indeed";

    cout << "The short string is " << small.length() << " characters." << endl;
    cout << "The long string is " << large.length() << " characters." << endl;

    return 0;
}
```

Output:

```
The short string is 10 characters.
The long string is 48 characters.
```

Accessing individual characters

Using square brackets [ ], you can access individual characters within a string as if it's a **char** array. Positions within a string type variable **str** are numbered from **0** through **str.length( ) - 1** (like array indices).

You can **read** and **write** to characters within a string using [ ]. For example, see the code sample below:

```
string test;
test = "I am Q the omnipot3nt";

char ch = test[5];      // ch is 'Q'
test[18] = 'e';         // we correct misspelling of omnipotent

cout << test << endl;
cout << "ch = " << ch << endl;
```

Output:

```
I am Q the omnipotent
ch = Q
```

Be careful not to access positions outside the bounds of the string. The square bracket operator [ ] is not **range-checked** and thus reading from or writing to an out-of-bounds index tends to produce difficult-to-track-down errors.
There is an alternate member function ***at (int index)*** that retrieves the character at a position with the benefit of built-in range-checking, but it's used much less often.

Passing, returning, assigning strings

C++ strings are designed to behave like ordinary primitive data types with regard to assignment. Assigning one string to another makes a deep copy of the character sequence. For example,

```
string str1 = "hello";
string str2 = str1;  // makes a new copy
str1[0]  = 'y';      // changes str1, but not str2
```

Passing and returning strings from functions clones/copies the string (or is passed by value). If you change a string parameter **within a function**, changes **are not seen** in the calling function unless you have specifically passed the string **by reference** (e.g. using the **&**).

Comparing two strings

You can compare two strings for equality using the **==** and **!=** operators. An example code is shown below:

```
string myName = "Neal";
while (true) {
   cout << "Enter your name (or 'quit' to exit): ";
   string userName = getLine();
   if (userName == "Julie") {
      cout << "Hi, Julie!  Welcome back!" << endl;
   } else if (userName == "quit") {
      // user is sick of entering names, so let's quit
      cout << endl;
      break;
   } else if (userName != myName) {
      // user did not enter quit, Julie, or Neal
      cout << "Hello, " << userName << endl;
   } else {
      cout << "Oh, it's you, " << myName << endl;
   }
}
```

Output:

```
Enter your name (or 'quit' to exit): Neal
Oh, it's you, Neal
Enter your name (or 'quit' to exit): Julie
Hi, Julie!  Welcome back!
Enter your name (or 'quit' to exit): Leland
Hello, Leland
Enter your name (or 'quit' to exit): quit
```

You can also use other comparison operators like **<**, **<=**, **>**, and **>=** to compare strings. These operators compare strings **lexicographically**, character by character and are case-sensitive. The following comparisons all evaluate to **true** (the result after comparison is **true**):

"A" < "B", "App" < "Apple", "help" > "hello", "Apple" < "apple"

The last one might be a bit confusing, but the ASCII value for 'A' is 65, and comes before 'a', whose ASCII value is 97. So "Apple" comes before "apple" (or, for that matter, any other word that starts with a lower-case letter).

Appending to a string

Suppose you have two strings, **s1** and **s2** and you want to create a new string of their concatenation. Conveniently, you can just write **s1 + s2**, and you'll get the result you'd expect.

Similarly, if you want to append to the end of string, you can use the **+=** operator. You can append either another string or a single character to the end of a string. For example,

```
string firstname = "Leland";
string lastname = " Stanford";

string fullname = firstname + lastname; // concat the two strings
fullname += ", Jr";     // append another string
fullname += '.';        // append a single char

cout << firstname << lastname << endl;
cout << fullname << endl;
```

Output:

```
Leland Stanford
Leland Stanford, Jr.
```

Searching within a string

The string member function **find( )** is used to search within a string for a particular string or character. A sample usage such as **str.find(key)** searches the **key** (the string or character value passed as argument) from the string **str**. The parameter **key** can either be a string or a character. The return value is either the starting position where the **key** was found or the constant **string::npos** that indicates the key was not found.

Example:

```
string sentence = "Yes, we went to Gates after we left the dorm.";

int firstWe = sentence.find("we"); // finds the first "we"
int secondWe = sentence.find("we", firstWe + 1); // finds "we" in "went"
int thirdWe = sentence.find("we", secondWe + 1); // finds the last "we"
int gPos = sentence.find('G');
int zPos = sentence.find('Z'); // returns string::npos

cout << "First we: " << firstWe << endl;
cout << "Second we: " << secondWe << endl;
cout << "Third we: " << thirdWe << endl;

cout << "Is G there? ";

if (gPos != string::npos) {
    cout << "Yes" << endl;
}
else {
    cout << "No" << endl;
}

cout << "Is Z there? ";

if (wPos != string::npos) {
    cout << "Yes" << endl;
}
else {
    cout << "No" << endl;
}
```

Output:

```
First we: 5
Second we: 8
Third we: 28
Is G there? Yes!
Is Z there? No!
```

Extracting substrings

Sometimes you would like to create new strings by extracting portions of a larger one. The **substr( )** member function creates substrings from pieces of the receiver string.

You specify the **starting position** and **the number of characters** (to extract from the original/receiver string). For example, **str.substr(start, length)** returns a new string consisting of the characters from **str** starting at the position **start** and continuing for **length** characters.

Invoking this member function does not change the receiver string, as it makes a new string with a copy of the characters specified. See the example code below:

```
string oldSentence;
oldSentence = "The quick brown fox jumped WAY over the lazy dog";
int len = oldSentence.length();
cout << "Original sentence: " << oldSentence << endl;
int found = oldSentence.find("WAY ");
string newSentence = oldSentence.substr(0, found);
cout << "Modified sentence: " << newSentence << endl;
newSentence += oldSentence.substr(found + 4);
cout << "Completed sentence: " << newSentence << endl;
```

Output:

```
Original sentence: The quick brown fox jumped WAY over the lazy dog
Modified sentence: The quick brown fox jumped
Completed sentence: The quick brown fox jumped over the lazy dog
```

Exercises

1. Write a C++ program where you should ask the use to input a date in the format DD/MM/YYYY (like **26/01/2022**) where day, month, and year are separated by a slash. Then, print day or date, month, and year on a separate line (like 26 should be printed on one line, 01 on the second line, and 2022 on the third line).
2. Create a user-defined function that should take one string parameter and it should return an integer value: **the count or number of *words* in that string**. Then, call it in the main ( ), where you should ask the user to enter any string value (can be an entire line), pass that value to the function being called, and display the returned value.
3. Write a C++ program that should ask the user to input a phone number where the network and number are separated by a dash "-" like 0300-1234567.
   Then, print which network is used in that number. For simplicity, assume the following codes are used by the respective networks:

Code	Network Name
0300	Mobilink
0333	Ufone
0315	Zong
0345	Telenor

4. Write a C++ program where you should ask the user to input his/her first and last names and create/print an **@iba-suk.edu.pk** email address with the entered first name and last name. For example, if the first name is **riaz** and last name is **ali**, then the program should make and print **riaz.ali@iba-suk.edu.pk** email address.
5. Create your own user defined function **my_lower( )** that takes one string parameter and return a string. The function should convert all the character in the string passed through arguments to **lowercase**. Then, call that function in the main ( ) function where you should ask the user to input a string, pass that string to the function created by you, and display the result returned by your function.
6. Create your own user defined function **my_upper( )** that takes one string parameter and return a string. The function should convert all the character in the string passed through arguments to **uppercase**. Then, call that function in the main ( ) function where you should ask the user to input a string, pass that string to the function created by you, and display the result returned by your function.
7. Write a program to check if a string (input by the user) is **palindrome** or not. A palindrome string is the one that reads the same if you read from backward or from forward, like **madam**, **radar**, **civic**, **anna**, **bob**, **refer** and so on.