Discipline Courses-I
Semester-I
Paper: Programming Fundamentals
Unit-V

Lesson: String Handling

Lesson Developer: Rakhi Saxena
College/Department: Deshbandhu College, University of Delhi

Table of Contents

2.1 String Handling -I

Many times when you write a program you will need to take in user input that is textual in nature such as – customer's name or address. For this you will need to work with strings. A string is nothing but a sequence of characters. It can be either of fixed length or of variable length.

In this chapter you will learn to create and manipulate strings in your programs.

2.1.1 <u>**Learning Objectives**</u>

After reading this chapter you should be able to:

- Use strings in a program.
    - Define and differentiate between C style strings and C style strings.
    - Declare and initialize C style strings.
    - Use string functions defined in <cstring>
    - Work with char arrays to find string length, copy strings, concatenate strings and compare strings.

2.1.2 String Concepts

A ***string*** is any finite, contiguous sequence of characters (letters, numerals, symbols and punctuation marks). An important characteristic of each string is its ***length***, which is the number of characters in it. The length can be any natural number (i.e., zero or any positive integer). An ***empty string*** is a string containing no characters and thus having a length of zero. A ***substring*** is any contiguous sequence of characters in a string.

You have already seen string literals such as "Hello World" – a sequence of characters enclosed in double quotes.

There are two options in C++ for working with trings:

- **C Style Strings** – These are strings that are treated as null terminated character arrays. The advantages of C Style strings are that they:
    - Offer a high level of efficiency
    - Give the programmer detailed control over string operations
- **C++ Style Strings** – These are string objects that are instances of the string class defined in the standard library (std::string). The advantages that C++ Style strings offer are:
    - Consistency: A string class defines a new data type that encapsulates data and operations that can be performed on strings.
    - Safety: Unlike C style strings where array out of bounds errors can occur, C++ style strings will not have such problems.
    - Convenience: Standard operators (such as =, +, and so on) can be used for string manipulation.

2.1.3 C Style Strings

C style strings use arrays to represent strings. To declare a C style string, we just need to declare a char array. For example,

```
char   name[20];
```

The char array *name* can store up to 20 elements of the type char. We can store strings that can be accommodated in 20 characters, such as "Ravi" or "Rajaravivarman" in the array. Since the char array can hold shorter sequences than its total length, a special character is used to signal the end of the valid sequence - the null character ('\0') that has ASCII code 0.

R	A	v	i	\0															

R	a	j	a	R	a	v	i	v	a	r	M	a	n	\0					

The size of the array thus needs to be one more than the longest string that the variable should hold to accommodate the null character. The values after '\0' in the array are undetermined values. Just like normal arrays, once an array is declared to be a particular size, the size can not be changed.

Value addition: Style Tip

Heading text C Style Strings

1. C style strings are fixed length char arrays. If you try to insert characters more than the length of the array, you will overwrite the null terminator and the runtime environment will not know where the string ends. If you try to print a string with no null terminator, you'll not only get the string, you'll also get everything in the adjacent memory slots until you happen to hit a 0. Make sure that you do not write in more characters than the length of the char array.
2. It is also a good idea to declare strings with more than enough space at the time of declaration if you are going to be using it for user input, say char name[255]. Otherwise, if the user were to enter more characters than the array could hold, you would get a buffer overflow. A buffer overflow results in other memory being overwritten which usually causes a program to crash.
3. When declaring strings initialized with string literals, it is always a good idea to use [] and let the compiler calculate the size of the array. That way if you change the string later, you won't have to manually adjust the size.

Source: Self Made

2.1.3.1 Initializing

C style strings can be initialized when they are declared – by assigning values to array elements or with string literals.

```
char name[] = {'R', 'a', 'v', 'i'};
char name[] = "Ravi";
```

Both these declarations declare an array of 5 elements of type char initialized with the characters that form the word "Ravi" plus a null character '\0' at the end.

Note: C style strings follow all the same rules as arrays. This means you can initialize the string upon creation, but you can not assign values to it using the assignment operator after that! For example, once we have declared the *name* array as

```
char name[] = "Rajaram";
```

we cannot write any statement like the following:

```
name = "Ranjan";     //is incorrect
name[] = "Ranjan";  //is incorrect
name = {'R', 'a', 'n', 'j', 'a', 'n'};//is incorrect
name[3] = 'w'; //is incorrect
```

2.1.3.2 Input and Output

Both the insertion (>>) and the extraction (<<) operators support null-terminated sequences as valid containers for sequences of characters, so they can be used directly to extract strings of characters from cin or to insert them into cout. For example, you can write code as below:

```
char  name[20];
cout << "Enter your name";
cin >> name;
cout << "Greetings " << name << "!" << endl;
```

Using cin to input a string works – it will read user input with spaces but it will terminate the string after it reads the first space. If blank spaces in user input are permitted, it is a better idea to use the cin.getline() function. This allows you to read in an entire string, even if it includes whitespace. You can use getline() as below:

```
cin.getline(name,20);
```

This call to cin.getline() will read up to 20 characters into *name* (leaving room for the null terminator). Using getline() also ensures that buffer overflow will not occur as any excess characters entered by the user will be discarded.

Value addition: Did You Know?

Heading text Working with pointers (C Style Strings)

A pointer to character can also be used to create and manipulate a C style string. For example, using the new operator you can allocate contiguous space for a 20 character array.

```
char *new_name;
new_name = new char[20];
```

To input and display the string, use

```
cin.getline(new_name, 20 );
cout << new_name;
```

To count the number of characters in the string, use

```
for (char *c = str; *c != '\0'; c++)
    length++;
```

To free all the memory locations occupied by the char pointer, use

```
delete[] new_name.
```

Source: Self Made

2.1.3.3 Manipulating

C++ provides many functions (contained in the <**cstring**> header) to manipulate C-style strings. The following table shows some string manipulation functions:

Function	**Purpose**
char ***strcpy**(char *s1, char * s2)	Copies a string to another - copies string s2 to string s1 - The contents of s1 after strcpy will be exactly the same as s2

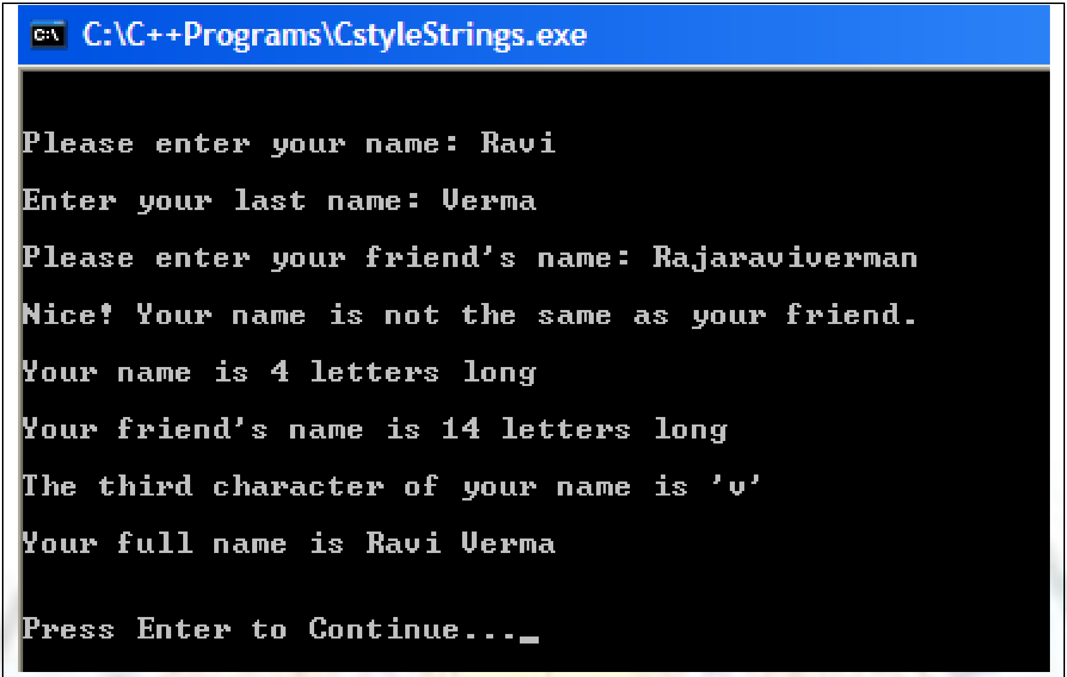
char ***strcpy**(char *s1, char * s2, int n)	Copies n characters of string s2 to string s1 - n should be less than the size of s1 or the write could still go beyond the bounds of the array
char ***strcat**(char *s1, char * s2)	Appends one string to another - concatenates s2 at the end of s1
char ***strncat**(char *s1, char * s2, int n)	Appends one string to another (with buffer length check) - takes first n characters of string s2 and appends them to string s1 - returns a pointer to the concatenated string
int **strcmp**(const char *s1, const char * s2)	Compares strings s1 and s2 (returns 0 if equal, less than 0 if s1<s2 and greater than 0 if s1>s2) – strcmp is case sensitive
int **strncmp**(char *s1, char * s2, int n)	Compares first n characters of string s1 with first n characters of string s2 (returns 0 if equal)
int **strlen**(char *s1, int len)	Returns the length of a string s1(excluding the null terminator)
char ***strchr**(*char *s1, char ch)	Finds a character in a string - returns the pointer at the first occurrence of ch
char ***strstr**(char *s1, char * s2)	Finds substring s2 in s1 - returns a pointer at the first occurrence of s1

Note: In some implementations the <cstring> library may be automatically included when you include other libraries such as the <iostream> library.

Value addition: Source Code

Heading text CStyleStrings

```
/* This program demonstrates usage of C Style Strings */

#include <iostream>
#include <cstring>
using namespace std;

int main()
{
  char name[50];
  char lastname[50];
  char fullname[100];
  char *new_name = new char[50];
  int count = 0;

  cout<<"\n\nPlease enter your name: ";
  cin.getline(name, 50 );
  cout<<"\nEnter your last name: ";
  cin.getline ( lastname, 50 );
  cout<<"\nPlease enter your friend's name: ";
  cin.getline(new_name, 50 );

  if ( strcmp (name, new_name) == 0 ) // Equal strings
    cout<<"\nCool! You and your friend have the same name.\n";
  else                                // Not equal
    cout<<"\nNice! Your name is not the same as your friend.\n";
  // Find the length of your name
  cout<<"\nYour name is "<< strlen ( name ) <<" letters long\n";

  // Find the length of your friend's name using pointer to array
  for ( char *c = new_name; *c != '\0'; c++)
      count++;
  cout<<"\nYour friend's name is "<< count <<" letters long\n";

  //Find a character in your name with array indexing
  cout <<"\nThe third character of your name is '" <<name[2]<<"'\n";

  fullname[0] = '\0';           // strcat searches for '\0' to cat after
  strcat ( fullname, name );     // Copy name into full name
  strcat ( fullname, " " );      // We want to separate the names by a space
  strcat ( fullname, lastname ); // Copy lastname onto the end of fullname
  cout<<"\nYour full name is "<< fullname <<"\n";
  cout<< "\n\nPress Enter to Continue...";
  getchar();
  return 0;
}
```



C:\C++Programs\CstyleStrings.exe

```
Please enter your name: Ravi
Enter your last name: Verma
Please enter your friend's name: Rajaraviverman
Nice! Your name is not the same as your friend.
Your name is 4 letters long
Your friend's name is 14 letters long
The third character of your name is 'v'
Your full name is Ravi Verma

Press Enter to Continue..._
```

Source: Self Made

2.1.4 <u>Working with char Arrays</u>

String manipulation can be conducted by manipulating the character sequence directly. We can write our own functions similar to those defined in <cstring> or others depending upon the requirement. In this section we will write functions to determine the length of strings, copy a string to another, concatenate two strings and compare two strings.

The function strlen() counts the number of bytes it finds in an array of characters. We can use a pointer to character which is initialized to point to the first character of the string. Then we increment the pointer and the length of the string until a terminating null is found.

```
int    strlen(char *str){
  int length=0;
  for (char *c = str; *c++;)          //loop runs until *c == '\0'
     length++;
  return length;
}
```

The function strcpy() copies a source char array to a destination char array by copying individual characters from the destination to the source until a terminating null is found in the destination string. The null character is also copied into the source string.

```
char *    strcpy(char *dest, const char *src){
  char *save = dest;                    //save a pointer to the beginning of the dest
  while (*dest++ = *src++);         //copy a char from dest to src until *dest == '\0'
  return (save);
```

```
        }
```

Following the same principle, the function strcat() locates the end of a sequence of characters, and copies another sequence of characters onto the end of the first sequence, thus concatenating (merging) two strings. Note that the second loop decrements the pointer to point just before the null character in the source string – the null is overwritten with the first character of the destination string.

```
char *   strcat(char *s, char *t){
    char *save;
    for (save = s; *s++; ) ;             // locate the end of string s
    for (--s; *s++ = *t++; ) ;           // copy string t to end of s
    return save;
}
```

The function strcmp() compares a string to another character by character until either a mismatch is found or until a null is found in both the strings. If both strings match upto the terminating null, a 0 is returned to indicate strings are equal. If the strings are not equal, the function returns a lexical difference of the two strings (s and t) passed as parameters. A positive value means that s would be after t in a dictionary. A negative value means that s would be before t in a dictionary.

```
int    strcmp(char *s, char *t){
   while (*s == *t++)                //run the loop until a mismatch is found
     if (!*s++)
       return 0;                     // strings are equal - return 0
   return s[0]-t[-1];               // strings are unequal-return the lexical difference
}
```

The function strncmp() find a substring within another. The source string is s1 and the substring to be searched is the first n characters from string s2.

```
int   strncmp(char *s1, char *s2, int n)
{
     if(n == 0) return 0;
     do
    {
           if(*s1 != *s2 ++) return *s1 - *-- s2;
           if(*s1 ++ == 0) break;
     }
     while (-- n != 0);
     return 0;
}
```

2.2 String Handling -II

In the last section you learnt to work with C style strings. In this section you will learn to work with C++ style strings.

2.2.1 **<u>Learning Objectives</u>**

After reading this chapter you should be able to:

- Use C++ style strings in a program.

- Employ various constructors to create string objects.
- Employ arithmetic and relational operators for string objects to concatenate, assign, and compare strings.
- Use string manipulation functions defined in the string class.

2.2.2 <u>C++ Style Strings</u>

C-style strings have many shortcomings, mainly, that you have to do all the memory management (allocate and free space for the array) yourself. There are no checks for buffer overflows. C++ strings provide a safer and easier way to work with strings than C style strings. The string class allows you to create variable length string objects – those that grow and shrink depending on the size of the text. You can assign strings to string objects using the assignment operator and they will automatically resize to be as large or small as needed.

C++ provides a string class that can be instantiated to create string objects. To use the string class, include the header:

```
#include <string>
using namespace std;
```

You can accept input for string objects using cin and display them with cout.

```
string name;
cout << “Enter your name”;
cin >> name;
cout << “Greetings “ << name << “!” << endl;
```

The getline() function can be used to read an entire string in, even if it includes whitespace.

```
getline(cin, name);
```

The string class provides a number of functions to assign, compare, and modify strings. You will learn more about C++ style strings in the next section.

Value addition: Did you Know?
Heading text C++ style strings
• These are the two classes that you will actually use when using the string class. std::string is used for standard ascii (utf-8) strings. std::wstring is used for wide-character/unicode (utf-16) strings. There is no built-in class for utf-32 strings (though you should be able to extend your own from basic_string<> if you need one). • Sequences of characters stored in char arrays (C style strings) can easily be converted into string objects (C++ style strings) by using the assignment operator: string mystring; char another_string[] = “Hello World”; mystring = another_string;
Source: Self Made

2.2.1 <u>String Constructors</u>

The string class provides a number of constructors that can be used to create strings. The following table shows some string constructors and their usage.

Constructor	**Purpose**	**Usage**	**Output**

string()	Default Constructor - Creates an empty string	string str;
		
		cout << str;	

string(const string& str)	Copy Constructor - Creates a new string from another string passed as parameter	string str1("hello world");
		string str2(str1);
		
		cout << str2;
			hello world

string(const string& str, size_t index)
 	
string(const string& str, size_t index, size_t length)
	Creates a new string that contains at most *length* characters from *str*, starting with *index*.
	
	If no length is supplied, all characters starting from index will be used.
		string str1("hello world");
		string str2(str1,3);
		string str3(str1,3,5);
		
		cout << str2<<endl<<str3;
			lo world
			lo wo

string(const char *str)
 	
string(const char *str, size_t length)
	Creates a new string from the C-style string str, up to but not including the NULL terminator.
	
	If length is supplied, creates a new string from the first length chars of str.
		string str1("hello world");
		string str2("hello world",5);
		
		cout << str1<<endl<<str2;
			hello world
			hello

string(size_t n, char ch)	Creates a new string initialized by n occurrences of the character ch.	string str(5,'C');
		cout << str;
			CCCCC

The string class destructor destroys the string and frees the memory. The destructor is automatically invoked when the object goes out of scope.

2.2.2 String Operators

The C++ arithmetic and relational operators are overloaded by the string class. The following table shows some operators and their usage. Assume the following declarations and initial values: string str1("Good"), str2("Morning"), string str3,str4;

Operator	Purpose	Usage	Output
<<	Input	cin << str3;	
>>	Output	cout >> str2;	Good

+	Concatenation	str3 = str1 + str2;
		str4 = str1 + "Night";
		cout <<str3<<endl<<str4;
			GoodMorning
			GoodNight

=	Assignment	str3 = str1;
		str4 = “Bye”;
		cout <<str3<<endl<<str4;
			Good
			Bye

+=	Append	str3 = "Nice";
		str3 += str2;
		cout << str3;
			NiceMorning

>, >=, <, <=, ==, !=	Comparison	if (str1 < str2) cout << "s1 is larger";	s1 is larger

[]	Subscript	cout << str2[4]<<endl;
		str1[0] = 'Z';
		cout << str1;
			i
			Zood

2.2.3 String Manipulation Functions

The string class has a number of functions to manipulate strings. Most of these functions are overloaded and you can use them in many ways. The following table summarizes some important string class member functions and shows some ways in which they can be used.

Function	Purpose	Usage	Output

append()	Append characters and strings onto a string	string str1("Hello");
		
		str1.append("!");
		cout << str1<<endl;
		str1.append(5,'!');
		cout << str1;
			Hello!
			Hello!!!!!!

assign()	Give a string values from strings of characters and other C++ strings	string str1;
		string str2 = "Cats and Dogs";
		
		str1.assign(str2, 5, 3);
		cout << str1 << endl;
			and

at()	Returns the character at a specific location	string str("Morning");
		cout << str.at(5);
			n

clear()	Removes all characters from the string	string str("Morning");
		str.clear();
		cout << str;	

compare()	Compares two strings (returns 0 if equal, less than 0 if str1 < str2 and greater than 0 if str1 > str2)	string str1("Good");
		string str2("Morning");
		
		cout << str1.compare(str2);
			-1

copy()	Copies characters from a string into a char array	string str1 = "Good Bye";
		char *str2;
		str1.copy(str2, 4);
		str2[4] = '\0';
			Good

		cout << str2 << endl;	

empty()	Returns true(1) if the string has no characters, false(0) otherwise	string s1;
		string s2("");
		string s3("Hello");
		cout << s1.empty() << endl;
		cout << s2.empty() << endl;
		cout << s3.empty() << endl;
			1
			1
			0

erase()	Removes characters from a string from a start index up to a last index	string str("Good Morning");
		str.erase(3,7);
		cout << str;
			Goong

find()	Find characters in the string starting from an index	string str("Good Morning");
		cout << "Found Morn at ";
		cout << str.find("Morn", 0);
			Found Morn at 5

insert()	Insert characters into a string at the specified index	string str("Good Morning");
		str.insert(2,"-");
		cout << str;
			Go-od Morning

length()	Returns the length of the string	string str("Good Morning");
		cout << str.length();
			12

replace()	Replace characters in the string from a start index up to a last index	string str1 = "Good Morning";
		string str2 = "Night";
		str1.replace(5, str1.length(), str2);
		cout << str1 << endl;
			Good Night

resize()	Change the size of the string	string str("Good");
		cout << str.length()<<endl;
		cout <<str << "*"<<endl;
		str.resize(10);
		cout << str.length()<<endl;
		cout <<str << "*"<<endl;
			12
			Good*
			20
			Good *

size()	Returns the number of characters in the string	string str("Good Morning");
		cout << str.length();
			12

substr()	Returns a certain substring from a start index up to a last index	string str("Good Morning");
		string sub = str.substr(5,4);
		cout << sub;
			Morn

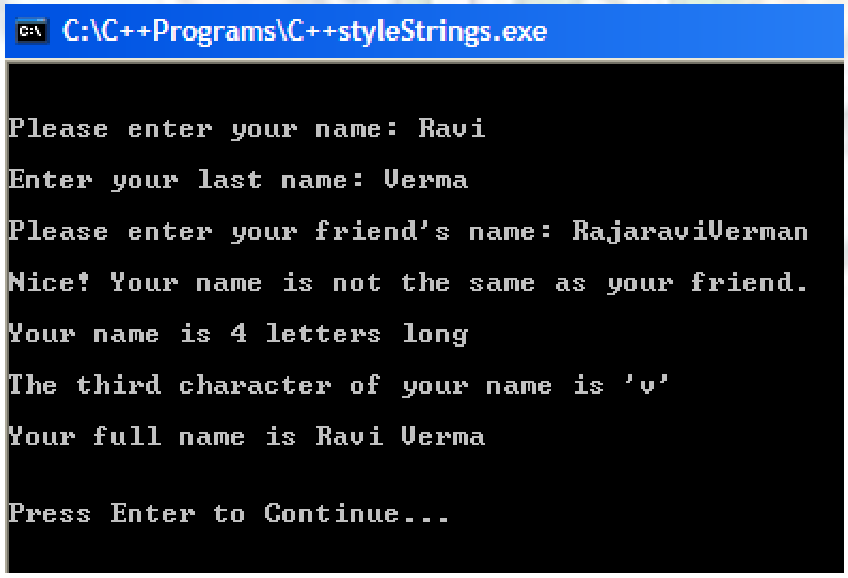
swap()	Swap the contents of the string with another	string str1("Good");
		string str2("Morning");
		str1.swap(str2);
		cout << str1 << " " << str2;
			Morning Good

Value addition: Source Code

Heading text C++StyleStrings

```
 /* This program demonstrates usage of C++ Style Strings */

#include <iostream>
#include <cstring>
```

```
using namespace std;

int main()
{
  string name;
  string lastname;
  string fullname;
  string new_name;

  cout<<"\n\nPlease enter your name: ";
  getline(cin, name);
  cout<<"\nEnter your last name: ";
  getline (cin, lastname);
  cout<<"\nPlease enter your friend's name: ";
  getline(cin,new_name);

  if ( name.compare(new_name) == 0 ) // Equal strings
    cout<<"\nCool! You and your friend have the same name.\n";
  else                                    // Not equal
    cout<<"\nNice! Your name is not the same as your friend.\n";
  // Find the length of your name
  cout<<"\nYour name is "<< name.length() <<" letters long\n";

  //Find a character in your name with array indexing
  cout <<"\nThe third character of your name is '" <<name[2]<<"'\n";

  fullname = name + " " + lastname; // We want to separate the names by a space
  cout<<"\nYour full name is "<< fullname <<"\n";

  cout<< "\n\nPress Enter to Continue...";
  getchar();
  return 0;
}
```



```
Please enter your name: Ravi

Enter your last name: Verma

Please enter your friend's name: RajaraviVerman

Nice! Your name is not the same as your friend.

Your name is 4 letters long

The third character of your name is 'v'

Your full name is Ravi Verma


Press Enter to Continue...
```

Source: Self Made

Value addition: Did you Know?

Heading text – Comparison of C style string and C++ style string functions

	C Style Strings	**C++ Style Strings**	

Library to include	#include <cstring>
	using namespace std;
		#include <string>	

Declaring string variable/object	char str[20];	string str;	

Initializing string variable/object	char str1[11] = "Hello World";
	char str2[] = "Good Night";
	char str3[] = {'O', 'K', '\0'};
		string str1("Hello World");
		string str2 = "Good Night";
		string str3("OK");	

Assigning	Not possible after creation	str2 = "Good Morning";
		str3 = str2;	

Concatenating	strcat(str1,str2);
	strcpy(str3, strcat(str1,str2));
		str1 += str2;
		str3 = str1 + str2	

Copying	strcpy(str, "Hello!");
	strcpy(str, otherString);
		str = "Hello";
		str = otherString;	

Accessing a single character	str[index]	str1[index] str.at[index] str[index,count]	
Comparing	strcmp(s1,s2);	s1< s2 s1 <= s2 s1 == s2	s1 > s2 s1 >= s2 s1 != s2
Finding Length	strlength(str);	str.length();	

String Output	cout << str;
	cout << setw(width) << str;
		cout << str;
		cout << setw(width) << str;	

String Input	cin >> str;
	cin.getline(str, n);
		cin >> str;
		getline(cin, str);
		getline(cin, str,n);	

Source:

2.3 String Handling - III

Now that you are familiar with C/C++ style strings and the operations that can be performed on them, we will build our own custom StringType class. This class will encapsulate the data and functions necessary to represent a character string and some common operations that can be applied to it. The advantage of such a class is that it gives you full control over how strings are implemented and manipulated. The purpose is not to develop an alternative to the string class of C++ but to teach you how any new type can be added into the C++ environment.

Learning to develop your own string class and thus creating your own data type will help you develop your object oriented programming skills.

2.3.1 Learning Objectives

After reading this chapter you should be able to:

- Create a custom string class.
- Define the data members required by the string class.
- Define and implement overloaded string class constructors and destructor.
- Overload the extraction and insertion operators for the string class.
- Overload the assignment, concatenation and relational operators for the string class.
- Overload the array subscripting operator to allow array index access to the string class.
- Implement member function compare to compare two string instances.
- Implement member function find to search a substring in a string instance.
- Implement member function to erase a specific sequence within a string object.
- Implement the member function to determine the length of the string instance.

2.3.2 Building a Custom String Class - StringType

We will build the string class around a regular C string (null terminated character array), but since the char array itself is declared as private, we're never allowed to access with it from the outside. Instead, we accomplish what we want by using the class' functions and operators.

Two data member variables are defined in the class.

```
class StringType {
    private:
            char * s;
            int size;

    …
};
```

The first one, s, is used to hold the actual characters of the string. It is dynamically allocated and reallocated (using new and delete). The string pointed to by s will be a normal null terminated character array. The other member variables, size, is an integer holding the number of characters in the string. It is always possible to compute the length of the char array each time it is needed, but since this value is needed very often, it is maintained as a data member of the class.

2.3.3 String Type Constructors and Destructor

Quite a few constructors are defined to declare StringType objects - A default constructor that creates an empty string, a copy constructor and a constructor that takes regular C strings.
The default constructor assigns a null terminated empty string to s.

```
StringType::StringType() {
  size = 1;       //size is at least 1 to accommodate the null character
  try {
     s = new char[size];
  } catch (bad_alloc ba) {
```

```
            cout << "Allocation Error \n";
            exit(1);
        }
        strcpy(s,"");  //create a null terminated string in s
    }
```

When a quoted string is used to initialize a StringType object, first the length of the string is determined using the strlen() function. This becomes the size of the StringType instance being created. Then sufficient memory is allocated for the char array and the quoted string is copied to the char array using the strcpy() function.

```
    StringType::StringType(char *str) {
        size = strlen(str) + 1;
        try {
            s = new char[size];
        } catch (bad_alloc ba) {
            cout << "Allocation Error \n";
            exit(1);
        }
        strcpy(s,str);
    }
```

The copy constructor is similar to the quoted string constructor – the string used to initialize the StringType instance is simply the char array of the object passed as parameter.

```
    StringType::StringType(const StringType &str) {
        size = str.size;
        try {
            s = new char[size];
        } catch (bad_alloc ba) {
                cout << "Allocation Error \n";
                exit(1);
        }
        strcpy(s,str.s);
    }
```

The destructor frees up the memory allocated to the char array of the instance.

```
        ~StringType() {delete [] s;}
```

2.3.4 StringType Assignment

The operator = is overloaded to enable quick and easy assignments. The set of assignment operators matches the constructors. Two types of assignments are possible - one to assign a StringType object to another and second to assign a string literal to a StringType instance. For example, the following assignments are possible

```
    StringType a,b(“Hello”);
    a = b;
    a = “Hello again”;
```

The functions work by first checking whether there is enough memory allocated to the object to perform the copying operation. If the memory allocated is less than that required, the old memory is freed and appropriate memory required is allocated. Then the string is copied into the char array of the instance and the result is returned. The functions that accomplish this are shown below:

```
    //Assign a StringType object to another
    StringType StringType::operator= (StringType str){
        StringType temp(str.s);

        if (str.size > size) {
          delete [] s;          //free previously allocated memory
          try {
              s = new char[str.size];
          }catch (bad_alloc ba) {
              cout << "Allocation Error \n";
              exit(1);
          }
          size = str.size
         }
         strcpy(s,str.s);
         strcpy(temp.s, str.s);
         return temp;
   }

   //Assign a string literal to a StringType object
   StringType StringType::operator= (char *str){
         int len = strlen(str) + 1;
         if (len > size) {
          delete [] s;          //free previously allocated memory
          try {
              s = new char[len];
          }catch (bad_alloc ba) {
              cout << "Allocation Error \n";
              exit(1);
          }
          size = len;
         }
         strcpy(s,str);
         return *this;
   }
```

2.3.5 StringType Input and Output

The StringType class overloads the insertion (<<) and extraction (>>) operators to provide convenient input and output. This allows the String type input and output to be performed as below:

```
        StringType name;
        cout<<"\n\nPlease enter your name: ";
        cin >> name;
        cout <<"Welcome " << name;
```

StringType output is simple - all that needs to be done is to display the char array data member. The function that accomplishes this is shown below.

```
   ostream& operator <<(ostream &out, StringType &str)
   {
     out << str.s;
     return out;
   }
```

Note that the parameter is passed by reference since it improves efficiency of our StringType class. Recall that whenever an object is passed as an argument to a function, a copy of that object is made. When the function terminates, the copy's destructor is called. However, when you pass by reference, no copy of the object is made. This means that when an object is passed by reference, neither is the copy constructor invoked to make a copy, nor is the destructor invoked to destroy the copy.

To take input for a StringType object, the problem is that we do not in advance how much memory to allocate for the char array data member. That is why, we first allocate a char buffer, temp, that can hold 255 characters. We then take input through getline() into this buffer. The next step is to copy the input string into the StringType instance (after first ensuring enough memory is allocated to the char array data member). The function that accomplishes this is shown below.

```
istream& operator >>(istream &in, StringType& str)
{
  char temp[255];
  int len;

  in.getline(temp,255);
  len = strlen(temp) + 1;

  if (len > str.size) {
     delete [] str.s;
     try {
        str.s = new char[len];
     }catch (bad_alloc ba) {
        cout << "Allocation Error \n";
        exit(1);
     }
     str.size = len;
  }
  strcpy(str.s, temp);
  return in;
}
```

2.3.6 <u>String Type Concatenation</u>

The operator + is overloaded to enable quick and easy concatenation. Two types of concatenations are possible - one to append a StringType instance to another and second to append a string literal to a StringType instance. The functions work by first checking whether there is enough memory allocated to the object to perform the concatenation operation. If the memory allocated is less than that required, the old memory is freed and appropriate memory required is allocated. Then the string is appended onto the char array of the instance and the result is returned. The functions that accomplish this are shown below:

```
//Overloaded concatenation operator + to append an object
StringType StringType::operator+ (StringType &str){
  int len;
  StringType temp;

  delete[] temp.s;
  len = strlen(str.s) + strlen(s) + 1;
  temp.size = len;
  try {
      temp.s = new char[len];
   }catch (bad_alloc ba) {
        cout << "Allocation Error \n";
        exit(1);
   }
   strcpy(temp.s,s);
   strcat(temp.s, str.s);
   return(temp);
}

//Overloaded concatenation operator + to append a string literal
StringType StringType::operator+ (char *str) {
  int len;
  StringType temp;

  delete[] temp.s;
  len = strlen(str) + strlen(s) + 1;
  temp.size = len;
  try {
      temp.s = new char[len];
   }catch (bad_alloc ba) {
        cout << "Allocation Error \n";
        exit(1);
   }
   strcpy(temp.s, s);
   strcat(temp.s, str);
   return(temp);
}
```

2.3.7 String Type Comparison

The relational operators are overloaded appropriately, to make it possible to compare a StringObject to another.

```
int operator==(StringType &str)  { return !strcmp(s, str.s);}
int operator!=(StringType &str)  { return strcmp(s, str.s);}
int operator <(StringType &str)  { return strcmp(s, str.s) < 0;}
int operator >(StringType &str)  { return strcmp(s, str.s) > 0;}
int operator <=(StringType &str) { return strcmp(s, str.s) <= 0;}
int operator >=(StringType &str) { return strcmp(s, str.s) >= 0;}
```

The relational operators are also overloaded, to make it possible to compare a StringObject to a quoted string.

```
int operator==(char* str)  { return !strcmp(s, str);}
int operator!=(char* str)  { return strcmp(s, str);}
int operator <(char* str)  { return strcmp(s, str) < 0;}
int operator >(char* str)  { return strcmp(s, str) > 0;}
int operator <=(char* str) { return strcmp(s, str) <= 0;}
int operator >=(char* str) { return strcmp(s, str) >= 0;}
```

2.3.8 String Type Utilities

Some member function utilities are also defined by the StringType class. The function compare() compares a StringType instance to another StringType instance and returns -1, 0 or 1 (regular strcmp() return values) to indicate the result. First the compare() function verifies whether either the invoking instance or the parameter are null and returns values appropriately. If neither is null, the strcmp is invoked on both their char array data members.

```
int StringType::compare(StringType& str)
{
        if (isNull() && !str.isNull())
                return 1;
        else if (!isNull() && str.isNull())
                return -1;
        else if (isNull() && str.isNull())
                return 0;
        return  strcmp(s, str.s);
}
```

The function find() locates a substring inside a StringType instance and returns its position in the parameter pos (a reference parameter). The substring to be searched is also passed as a StringType parameter. The find() function returns true if the substring is found, false otherwise.

```
bool StringType::find(StringType& str,int& pos)
{       if (isNull() ||str.isNull())
                return false;
        for (pos=0 ; pos<(size-str.size) ; pos++)
        {
          if (strncmp(&s[pos], str.s, str.size-1) == 0) //compare substring to s
                return true;                          // starting from first position ,
        }                                             //then second position and so on
        return false;
}
```

The function erase() removes a substring from within a StringType instance. The starting index and the number of characters are passed as parameters.

```
void StringType::erase(int pos, int count)
{
        if (pos > size || count==0)  // Start is outside string or nothing to delete
                return;
        if (pos + count > size)           // clamp count to string length; cannot
                count = (size - pos);    //delete more characters than string length

        // Characters are deleted by moving up the data
        for (int i=pos ; i<size-count ; i++)
                s[i] = s[i+count];

       size -= count;                   //change the size of the string
       s[size] ='\0';                   //insert the null character at the end of the string
}
```

2.3.9 StringType Subscripting

The [] operator returns a reference to the character at a specific index in the character array. Since it's a reference, the following code is perfectly legal:

```
StringType  str = StringType("Hello World");
str[0] = 'F';
```

The StringType class overloads the [] operator to return a reference to the required index. If the array index is out of bounds the first character is returned.

```
char &operator[](int i)     {
    if ( i >= 0 && i < size)
        return s[i];
    else
        return s[0];
}
```

Value addition: SourceCode

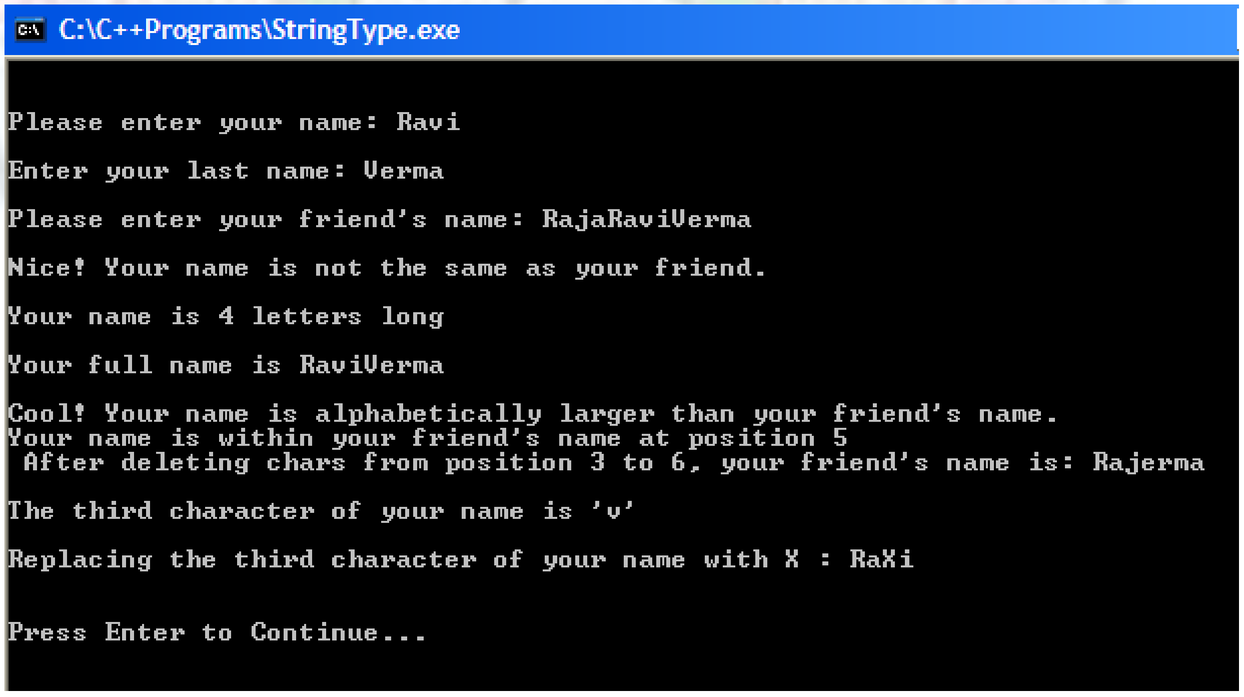
Heading text StringType Class

```
/* Custom String Class - Creates the class StringType that encapsulates
data and member functions needed to work with strings
Neither <cstring> nor <string> is used. All string operations are
performed with the StringType in main()*/

#include <iostream>
using namespace std;

class StringType {
   private:
      char * s;
      int size;
   public:
      StringType();                    //default constructor
      StringType(char *str);           //initailize with a string literal
      StringType(const StringType &s);  //copy constructor

      ~StringType() {delete [] s;}      //destructor

      friend ostream &operator <<(ostream &out, StringType &str);
      friend istream &operator >>(istream &in, StringType &str);

      StringType operator= (StringType str); //assign a object
      StringType operator= (char *str);       //assign a string literal

      StringType operator+ (StringType &str); //concatenate a object
      StringType operator+ (char *str);       //concatenate a string literal

      //relational operators between StringType objects
      int operator==(StringType &str)  { return !strcmp(s, str.s);}
      int operator!=(StringType &str)  { return strcmp(s, str.s);}
      int operator <(StringType &str)  { return strcmp(s, str.s) < 0;}
      int operator >(StringType &str)  { return strcmp(s, str.s) > 0;}
      int operator <=(StringType &str) { return strcmp(s, str.s) <= 0;}
      int operator >=(StringType &str) { return strcmp(s, str.s) >= 0;}
      //relational operators between StringType objects and literals
      int operator==(char* str)  { return !strcmp(s, str);}
      int operator!=(char* str)  { return strcmp(s, str);}
      int operator <(char* str)  { return strcmp(s, str) < 0;}
      int operator >(char* str)  { return strcmp(s, str) > 0;}
      int operator <=(char* str) { return strcmp(s, str) <= 0;}
      int operator >=(char* str) { return strcmp(s, str) >= 0;}

      //access through subscripting; return first character
      //if array index out of bounds
      char &operator[](int i)     { if ( i >= 0 && i < size)
                                     return s[i];
                                   else
                                     return s[0]; }

      int length() {return size-1; }      //returns length of StringType object
                                         //do not include null character in length

      bool isNull() { return (size == 1);}
                                        // Returns true if a NULL string is held by an instance.
           //String utility functions
     int compare(StringType &str);
     bool find(StringType& str, int &pos);
     void erase(int pos, int count);

            //helper functions for C style string operations
      int strlen(char *src);
      char * strcpy(char *dest, const char *src);
      char * strcat(char *str1, char * str2);
      int strcmp(char *str1, char *str2);
      int strncmp(char *s, char *find, int n);
      bool isNull(char *s) {return (s == NULL); }
};

//function to find length of a char array
int StringType::strlen(char *str){
   int length=0;
   for (char *c = str; *c++;)     //loop runs until *c == '\0'
       length++;
   return length;
}
//function to copy a source char array to a destination char array
char *StringType::strcpy(char *dest, const char *src){
   char *save = dest;
   while (*dest++ = *src++);   //copy a char from dest to src until *dest == '\0'
   return (save);
}
//function to compare a string to another
int StringType::strcmp(char *s, char *t){
    while (*s == *t++)
       if (!*s++)
          return 0;
    return s[0]-t[-1];
```

```
}
//function to concatenate a string to the end of another
char *StringType::strcat(char *s, char *t){
    char *save;
          for (save = s; *s++; ) ;  // locate the end of string s
          for (--s; *s++ = *t++; ) ; // copy string t to end of s
          return save;
}
//function to find a substring within another
int StringType::strncmp(char *s1, char *s2, int n)
{
 if(n == 0) return 0;
 do
 {
  if(*s1 != *s2 ++) return *s1 - *-- s2;
  if(*s1 ++ == 0) break;
 }
 while (-- n != 0);
 return 0;
}
//default constructor
StringType::StringType() {
   size = 1;       //size is at least 1 to accommodate the null character
   try {
       s = new char[size];
   } catch (bad_alloc ba) {
           cout << "Allocation Error \n";
           exit(1);
   }
   strcpy(s,"");  //create a null terminated string in s
}
//constructor to initialize with a string literal
StringType::StringType(char *str) {
   size = strlen(str) + 1;
   try {
       s = new char[size];
   } catch (bad_alloc ba) {
       cout << "Allocation Error \n";
       exit(1);
   }
   strcpy(s,str);
}
//copy constructor
StringType::StringType(const StringType &str) {
   size = str.size;
   try {
       s = new char[size];
   } catch (bad_alloc ba) {
           cout << "Allocation Error \n";
           exit(1);
   }
   strcpy(s,str.s);
}
```

```
//overloaded insertion operation <<
ostream& operator <<(ostream &out, StringType &str)
{
  out << str.s;
  return out;
}
//overloaded extraction operator >>
istream& operator >>(istream &in, StringType& str)
{
  char temp[255];
  int len;

  in.getline(temp,255);
  len = strlen(temp) + 1;

  if (len > str.size) {
     delete [] str.s;
     try {
        str.s = new char[len];
     }catch (bad_alloc ba) {
        cout << "Allocation Error \n";
        exit(1);
     }
     str.size = len;
  }
  strcpy(str.s, temp);
  return in;
}
//overloaded assignment operator = to assign a object
StringType StringType::operator= (StringType str){

    StringType temp(str.s);

    if (str.size > size) {
     delete [] s;          //free previously allocated memory
     try {
        s = new char[str.size];
     }catch (bad_alloc ba) {
        cout << "Allocation Error \n";
        exit(1);
     }
     size = str.size;
    }
    strcpy(s,str.s);
    strcpy(temp.s, str.s);
    return temp;
}
//overloaded assignemnt operator = to assign a string literal
StringType StringType::operator= (char *str){
    int len = strlen(str) + 1;
    if (len > size) {
     delete [] s;          //free previously allocated memory
     try {
        s = new char[len];
     }catch (bad_alloc ba) {
        cout << "Allocation Error \n";
        exit(1);
     }
     size = len;
    }
    strcpy(s,str);
    return *this;
}
//Overloaded concatenation operator + to append an object
StringType StringType::operator+ (StringType &str){
  int len;
  StringType temp;

  delete[] temp.s;
  len = strlen(str.s) + strlen(s) + 1;
  temp.size = len;
  try {
     temp.s = new char[len];
  }catch (bad_alloc ba) {
        cout << "Allocation Error \n";
        exit(1);
  }
  strcpy(temp.s,s);
  strcat(temp.s, str.s);
  return(temp);
}
//Overloaded concatenation operator + to append a string literal
StringType StringType::operator+ (char *str) {
  int len;
  StringType temp;

  delete[] temp.s;
  len = strlen(str) + strlen(s) + 1;
  temp.size = len;
  try {
     temp.s = new char[len];
  }catch (bad_alloc ba) {
        cout << "Allocation Error \n";
        exit(1);
  }
  strcpy(temp.s, s);
  strcat(temp.s, str);
  return(temp);
}
// Compares another string with this instance. Return values
//equals those of strcmp() -1, 0 or 1 (Less, Equal, Greater)
int StringType::compare(StringType& str)
{

                    if (isNull() && !str.isNull())
```

```
                        return 1;

                else if (!isNull() && str.isNull())

                        return -1;
                else if (isNull() && str.isNull())
                        return 0;
                return  strcmp(s, str.s);
}
// Finds a substring within this string.
// Returns true if found (position in pos).
bool StringType::find(StringType& str,int& pos)
{

                if (isNull() ||str.isNull())
                        return false;
                for (pos=0 ; pos<(size-str.size) ; pos++)
                {
         if (strncmp(&s[pos], str.s, str.size-1) == 0)
                                return true;
                }
                return false;
}
// Erases the specified number of characters,
// starting at the specified position.
void StringType::erase(int pos, int count)
{

                if (pos > size || count==0)   // Start is outside string
                        return;                //or nothing to delete
                if (pos + count > size)        // Clamp Count to string length
                        count = (size - pos);

                // Characters are deleted by moving up the data following
                //the region to delete.
                for (int i=pos ; i<size-count ; i++)
                        s[i] = s[i+count];
           size -= count;
                s[size] ='\0';
}

int main() {

  StringType name, lastname, fullname, new_name;
  int pos;

  cout<<"\n\nPlease enter your name: ";
  cin >> name;
  cout<<"\nEnter your last name: ";
  cin >> lastname;
  cout<<"\nPlease enter your friend's name: ";
  cin >> new_name;
  if ( name == new_name) // Equal strings
    cout<<"\nCool! You and your friend have the same name.\n";
  else                  // Not equal
    cout<<"\nNice! Your name is not the same as your friend.\n";
  cout<<"\nYour name is "<< name.length() <<" letters long\n";
  fullname = name + lastname;
  cout<<"\nYour full name is "<< fullname <<"\n";
  if ( name.compare(new_name) > 0)
    cout<<"\nCool! Your name is alphabetically larger than your friend's name.\n";
  else
    cout<<"\nNice! Your name is alphabetically smaller than your friend's name.\n";
  if (new_name.find(name, pos))
     cout << "Your name is within your friend's name at position" << pos+1 <<"\n ";
  else
     cout << "Your name is not contained within your friend's name\n";
  new_name.erase(3,6);
  cout << "After deleting chars from position 3 to 6";
  cout << "your friend's name is: " <<new_name<<endl;
  //Find a character in your name with array indexing
  cout <<"\nThe third character of your name is '" <<name[2]<<"'\n";
  name[2] = 'X';                //Replace third character
  cout <<"\nReplacing the third character of your name with X : " << name<<"\n";

  cout<< "\n\nPress Enter to Continue...";
  getchar();
  return 0;
}
```



```
Please enter your name: Ravi

Enter your last name: Verma

Please enter your friend's name: RajaRaviVerma

Nice! Your name is not the same as your friend.

Your name is 4 letters long

Your full name is RaviVerma

Cool! Your name is alphabetically larger than your friend's name.
Your name is within your friend's name at position 5
 After deleting chars from position 3 to 6, your friend's name is: Rajerma

The third character of your name is 'v'

Replacing the third character of your name with X : RaXi


Press Enter to Continue...
```

Source: Self Made

Summary

- A *string* is any finite, contiguous sequence of characters (letters, numerals, symbols and punctuation marks).
- The *length* of a string is the number of characters in it. The length can be any natural number (i.e., zero or any positive integer).
- An *empty string* is a string containing no characters and thus having a length of zero.
- A *substring* is any contiguous sequence of characters in a string.
- A string literal is a sequence of characters enclosed in double quotes.
- C style strings are strings that are treated as null terminated character arrays.
- The advantages of C style strings are that they offer a high level of efficiency and give the programmer detailed control over string operations
- C++ style strings are string objects that are instances of the string class defined in the standard library (std::string).
- The advantages that C++ style strings offer are consistency (a string class defines a new data type), safety (unlike C style strings where array out of bounds errors can occur, C++ style strings will not have such problems) and convenience (standard operators can be used for string manipulation).
- To declare a C style string, we just need to declare a char array. For example, char name[20] or char *name;
- Since a char array can hold shorter sequences than its total length, a special character is used to signal the end of the valid sequence - the null character ('\0') that has ASCII code 0.
- C style strings can be initialized when they are declared – by assigning values to array elements or with string literals.
- Both the operators, >> and << , support null-terminated sequences as valid containers for sequences of characters, so they can be used directly to extract strings of characters from cin or to insert them into cout.
- The getline() function allows you to read in an entire string, even if it includes whitespace.
- A pointer to character can also be used to create and manipulate a C style string.

- C++ provides many functions (contained in the <**cstring**> header) to manipulate C-style strings.
- C++ provides a string class that can be instantiated to create string objects. To use the string class, include the header:

  ```
  #include <string>
  using namespace std;
  ```
- You can accept input for string objects using cin and display them with cout.
- The string class allows you to create variable length string objects – those that grow and shrink depending on the size of the text.
- You can assign strings to string objects using the assignment operator and they will automatically resize to be as large or small as needed.
- The string class also provides a number of functions to assign, compare, and modify strings.
- We can build our own custom StringType class to encapsulate the data and functions necessary to represent a character string and operations that can be applied to it.

Exercises

1.1 What is the difference between char a[] = “string”; and char *p = “string”; ?

1.2 Are the following definitions valid? Why or why not?

```
string message = "Hello";
message = hello + ", world" + "!";
```

1.3 The following functions are all *intended* to check whether a string contains any lowercase letters, but at least some of them are wrong. For each function, describe what the function actually does. (Note: The function int islower (int c) checks if parameter *c* is a lowercase alphabetic letter. It returns true if *c* is a lowercase alphabetic letter, false otherwise).

-
  ```
  bool islowercase1(char *s){
    for(char *c = s; *c++;)
     if (islower(*c))
        return true;
     else
        return false;
  }
  ```

-
  ```
  bool islowercase2(char *s){
   for(char *c = s; *c++;)
     if (islower('c'))
        return true;
     else
        return false;
  }
  ```

-
  ```
  bool islowercase3(char *s){
   int flag;
   for(char *c = s; *c++;)
     flag = islower(*c);
   return flag;
  }
  ```

-
  ```
  bool islowercase4(char *s){
    int flag = false;
    for(char *c = s; *c++;)
       flag = flag || islower(*c);
  ```

```
      return false;
    }
```

-
  ```
  bool islowercase5(char *s){
     for(char *c = s; *c++;)
        if (!islower('c'))
           return true;
        else
           return false;
  }
  ```

1.4 Implement the following functions that work for strings as character arrays:

- char ***strncpy**(char *s1, char * s2, int n) : Copies one string to another (with buffer length check) - takes first n characters of string s2 and copies them to string s1 - returns a pointer to the copied string
- char ***strncat**(char *s1, char * s2, int n) : Appends one string to another (with buffer length check) - takes first n characters of string s2 and appends them to string s1 - returns a pointer to the concatenated string
- char ***strchr**(*char *s1, char ch) : Finds a character in a string - returns the pointer at the first occurrence of ch
- char ***strstr**(char *s1, char * s2): Finds substring s2 in s1 - returns a pointer at the first occurrence of s1
- char ***strrev(**char *s1**):** reverses the order of the characters in the given string s1. The ending null character (\0) remains in place. A pointer to the altered *string* is returned.

1.5 Implement the following functions that work for strings as string class instances:

- Function to accept a string as user input and determine of it is a palindrome. A palindrome is a string that reads the same forward and backward. (For example, “level” is a palindrome; “string” is not a palindrome)
- Function that prints out all the permutations of a string. For example, if “abc” is the input, the function should print abc, acb, bac, bca, cab, cba.
- Function that converts all characters of an input string to upper case characters. (You can use the library toupper and tolower functions.)

1.6 ROT13 is a weak form of encryption that involves “rotating” each letter in a word by 13 places. To rotate a letter means to shift it through the alphabet, wrapping around to the beginning if necessary, so ’A’ shifted by 3 is ’D’ and ’Z’ shifted by 1 is ’A’. Write a function called *rotate_word()* that takes a string and an integer as parameters, and that returns a new string that contains the letters from the original string “rotated” by the given amount. (For example, “cheer” rotated by 7 is “jolly” and “melon” rotated by -10 is “cubed”.)

1.7 How does parameter passing by reference improve the efficiency of the StringType class?

1.8 For the StringType class, implement the following:

- The function clear() that removes all characters from the string
- the function padRight(int n) to add extra blanks to the *right* end of a string to make it at least length n. If the string is already n characters or longer, do not change the string.
- two versions of the function insert(), one which inserts a character at a given location and one which inserts a complete string at a given location. (Remember to allocate memory for the added characters and to change the size of the object
- The function swap() that swaps the contents of the instance with another passed as parameter.
- Overload the += operator for the class.

Glossary

C style Strings: strings that are treated as null terminated character arrays.

C++ style Strings: string objects that are instances of the string class defined in the standard library (std::string).

Empty String: a string containing no characters and thus having a length of zero.

String: any finite, contiguous sequence of characters (letters, numerals, symbols and punctuation marks).

String Length: the number of characters in a string, the length can be any natural number (i.e., zero or any positive integer).

String Literal: a sequence of characters enclosed in double quotes.

Substring: any contiguous sequence of characters in a string.

References

1. Works Cited

2. Suggested Reading

1. B. A. Forouzan and R. F. Gilberg, Computer Science, A structured Approach using C++, Cengage Learning, 2004.
2. R.G. Dromey, How to solve it by Computer, Pearson Education
3. E. Balaguruswamy, Object Oriented Programming with C++ , 4th ed., Tata McGraw Hill
4. G.J. Bronson, A First Book of C++ From Here to There, 3rd ed., Cengage Learning.
5. Graham Seed, An Introduction to Object-Oriented Programming in C++, Springer
6. J. R. Hubbard, Programming with C++ (2nd ed.), Schaum's Outlines, Tata McGraw Hill
7. D S Malik, C++ Programming Language, First Indian Reprint 2009, Cengage Learning
8. R. Albert and T. Breedlove, C++: An Active Learning Approach, Jones and Bartlett India Ltd.

3. Web Links

1.1 http://anaturb.net/C/string_exapm.htm

1.2 http://cplus.about.com/od/learning1/ss/strings.htm

1.3 http://www.cplusplus.com/reference/string/string/

1.4 http://linuxsoftware.co.nz/cppstrings.html

1.5 http://www.macs.hw.ac.uk/~pjbk/pathways/cpp1/node183.html