MECH215
Lecture Notes 10

String Constants

A string constant is a sequence of 0 or more characters enclosed in double quotes. For example, the following program contains three different string constants:

```
#include <iostream>
// Note, we intentionally omitted the <string> library
using namespace std;

int main() {

  cout << "Hello" << endl;
  cout << "" << endl;
  cout << "Good\tbye" << endl;   // \t is a horizontal TAB

  return 0;
}
```

The individual characters composing a string are stored in SUCCESSIVE memory locations. String constants are TERMINATED with the special NULL character (the all 0s bit pattern which is specified with the \0 escape sequence. The null character is not included when counting the number of characters in a string.

Address	Memory Contents
1000	'H'
1001	'e'
1002	'l'
1003	'l'
1004	'o'
1005	'\0'
3001	'\0'
4000	'G'
4001	'o'
4002	'o'
4003	'd'
4004	'\t'
4005	'b'
4006	'y'
4007	'e'
4008	'\0'

Note that the address given in this figure are random. The important thing to take note of is that individual characters of a given string constant are stored in successive memory locations.

The string Class

The C++ language does not provide a fundamental data type for naming and storing strings. The library <string> provides a C++ class which provides a mechanism for declaring and assigning values to a string object and various functions used to manipulate string objects. If we wish to make use of these features, we must

```
#include <string>
```

at the beginning of the source code. In the above program, we are merely sending a string constant to the << operator, it is not necessary to #include <string>. Detailed documentation on the class can be found at http://www.cplusplus.com/reference/string/string/

Consider the following program which makes use of the features available in the <string> library:

```
// Author: Ted Obuchowicz and John Peach
// StringConstant.cpp

#include <iostream>
#include <string>

using namespace std;

int main() {

  string singer       = "Mick";
  string guitarPlayer = "Keith";
  string glimmerTwins = singer + " and " +  guitarPlayer;
  string husbandOfJerry;

  husbandOfJerry = singer;

  cout << singer         << endl;
  cout << guitarPlayer   << endl;
  cout << glimmerTwins   << endl;
  cout << husbandOfJerry << endl;

  return 0;
}
```

In C++, we can declare a string object in the following manner:

```
string someName;
```

For example, the two lines:

```
string singer       = "Mick";
string guitarPlayer = "Keith";
```

create two string objects called singer and guitarPlayer. A string object has a set of attributes associated with it, among these are:

- the characters composing the string
- the number of characters contained in the string object

Thus, we can visualize the two string objects singer and guitarPlayer in the following manner:

```
singer   ----------------
          'M' 'i' 'c' 'k'

          length : 4
         ---------------


guitarPlayer    ------------------
                'K' 'e' 'i' 't' 'h'

                 length : 5
                ------------------
```

Note that unlike string constants, the characters comprising a string object do not have the terminating '\0' character.

A string object may be assigned a value at the time of its declaration. This has been done for the two string objects called singer and guitarPlayer.

A string object may be formed from previously defined string objects by using the STRING CONCATENATION OPERATOR (+) as in the line:

```
string glimmerTwins = singer + " and " +  guitarPlayer;
```

A value can be ASSIGNED to a string object by using the assignment statement as in:

```
husbandOfJerry = singer;
```

Reading In A string From The Keyboard

Suppose we wanted to read in a string from the keyboard and assign it to a string object. Furthermore, suppose we are interested in entering a person's first and last name into a single string object. Our first attempt is to use the cin stream together with the insertion operator >> :

```
// Author: Ted Obuchowicz and John Peach
// ReadString.cpp

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

int main() {

  string name;
  cout << "Enter your name: ";
  cin >> name;

  cout << "Hello, " << name << endl;

  return 0;
}
```

Here is an interaction with the program:

```
Enter your name: John Peach
Hello, John
```

We enter the two words John Peach but only John is read into the string object name. What went wrong?

Actually, nothing went wrong. The reason for the behaviour is that the extraction operator skips and initial white space (blank) characters and keeps reading non-blank characters into its string operand.

The getline() string Library Function

The getline() function is used to read an entire line of input into a string object. When using the getline() function, we must specify three arguments:

1. The stream to read from (for example read characters from the cin stream)
2. The string object which is to receive the input
3. The character which specifies when to terminate input (we use the '\n' character if we want to read an entire line into a string object)

Here is a different version which now reads in John Peach into the string object name:

```
// Author: Ted Obuchowicz and John Peach
// ReadLine.cpp

#include <iostream>
#include <string>

using namespace std;

int main() {

  string name;
  cout << "Enter your name: ";
  getline(cin, name, '\n');

  cout << "Hello, " << name << endl;

  return 0;
}
```

When run, the output is:

```
Enter your name: John Peach
Hello, John Peach
```

More string Methods

The <string> library contains several useful methods (formally called functions) which can be used on string objects. The way in which we invoke these string functions is the following:

stringObjectName.methodName(argument list)

We briefly discuss some of the available methods found in the string library:

size()

The size method is used to obtain the number of characters contained in a string object. It is used in the following manner:

```
// Author: Ted Obuchowicz and John Peach
// StringSize.cpp

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

```
using namespace std;

int main() {

  string alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  int letterCount;

  letterCount = alphabet.size();
  cout << "The string " << alphabet << " has " << letterCount
       << " characters in it." << endl;

  return 0;
}
```

Here is the output:

```
The string ABCDEFGHIJKLMNOPQRSTUVWXYZ has 26 characters in it.
```

substr(int start, int length)

We can use the substr method to return a substring of a particular string. We specify the starting position of where the substring is to commence from with the first argument . The second argument specifies the number of characters which should make up the substring.

For example;

```
// Author: Ted Obuchowicz and John Peach
// SubstrDate.cpp

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

int main() {

  string date  = "September 23, 2012";
  string month = date.substr(0,9);
  string day   = date.substr(10, 2);
  string year  = date.substr(14, 4);

  cout << month << endl;
  cout << day << endl;
  cout << year << endl;

  return 0;
}
```

The program output is:

```
September
23
2012
```

NOTE: The position numbers of characters within a string object start at position 0 and not position 1.

find(string, int starting_position)

This method is used to find a particular substring within another string. If the substring is contained within the string, the find(0 method returns the substring's starting position within the string. The parameter starting_position specifies where to start looking for the substring.

The following program makes use of the find() and substring() methods to extract the first and second name from a name consisting of the of string of the form first_name*second_name and prints the two names in reverse order:

```
// Author: Ted Obuchowicz and John Peach
// StringFind.cpp

#include <iostream>
#include <string>

using namespace std;

int main() {

  string input = "Ted*Obuchowicz";
  string first, second;

  // get the length of the input string

  int howLong   = input.size();
  int firstStar = input.find("*", 0);

  first   = input.substr(0, firstStar );
  second  = input.substr(firstStar + 1, howLong - 1);

  cout << second << " " << first << endl;

  return 0;
}
```

The program output is:

```
Obuchowicz Ted
```

c_str()

The c_str() method of the string class will return a constant pointer to a cstring (null terminated) that is equivalent to the value of the string. Since, the pointer is constant, then we cannot modify the data in the the cstring. This is because the string class is called an immutable class. That means, once the data has been stored in it, it cannot be changed, mutated. To change the data we need to create a new object. If we wanted to change the text, we can access it an copy it to another cstring tha is not a constant.

```
// Author: John Peach
// StringCStr.cpp

#include <iostream>
#include <string>
#include <cstring>

using namespace std;

int main() {

  string input = "C++ has strings and cstrings";
  const char* readOnlyCString = input.c_str();
  char* writableCString = new char[input.size()];

  strncpy(writableCString, readOnlyCString, input.size());
  writableCString[20] = 'C';
  cout << input << endl;
  cout << readOnlyCString << endl;
  cout << writableCString << endl;

  return 0;
}
```

The program output is:

```
C++ has strings and cstrings
C++ has strings and cstrings
C++ has strings and Cstrings
```

Note that the array readOnlyCString is declared as a "const char*" and not a "char*". The function strncpy is used to copy the data from readOnlyCString to writableCString. The content of writableCString can be changed, while readOnlyCString cannot be.