# 2 The Standard string Class

- The string class allows the programmer to treat strings as a basic data type
  - No need to deal with the implementation as withC-strings
- The string class is defined in the string library and the names are in the standard namespace
  - To use the string class you need these lines:

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

## 2.1 Assignment of Strings

- Variables of type string can be assigned with the = operator
  - Example:

```
string s1, s2, s3;
:

s3 = s2;
```

- Quoted strings are type cast to type string
  - Example:

```
string s1 = "Hello cosc3000!";
```

## 2.2 Using + With strings

- Variables of type string can be concatenated with the + operator
  - Example:

```
string s1, s2, s3;
:

s3 = s1 + s2;
```

    - If **s3** is not large enough to contain **s1** + **s2**, more space is allocated

## 2.3 string Constructors

- The default string constructor initializes the string to the empty string
- Another string constructor takes a C-string argument
  - Example:

```
string phrase; // empty string
string noun("ants"); // a string version of "ants"
```

## 2.4 Mixing strings and C-strings

- It is natural to work with strings in the following manner

```
string phrase = "I love" + adjective + " " + noun + "!";
```

  - It is not so easy for C++!
    - It must either convert the null-terminated C-strings, such as "I love", to strings,
    - or it must use an overloaded + operator that works with strings and C-strings.

## 2.5 I/O With Class string

- The insertion operator << is used to output objects of type string
  - Example:
```
string s = "Hello cosc3000/6000!";
cout << s;
```
- The extraction operator >> can be used to input data for objects of type string
  - Example:
```
string s1;
cin >> s1;
```
    * >> skips whitespace and stops on encountering whitespace

### 2.5.1 getline and Type string

- A **getline** function exists to read entire lines into a string variable
  - This version of **getline** is not a member of the **istream** class, it is a non-member function
  - Syntax for using this **getline** is different than that used with **cin**: **cin.getline**(...)
- Syntax for using **getline** with string objects:
```
getline(Istream_Object, String_Object);
```
- This code demonstrates the use of **getline** withstring objects
```
string line;
cout "Enter a line of input:\n";
getline(cin, line);
cout << line << "END OF OUTPUT\n";
```
  - Output could be:

```
Enter some input:
Do be do to you!
Do be do to you!END OF OUTPUT
```

### 2.5.2 Another Version of getline

- The versions of **getline** we have seen, stop reading at the end of line marker '\n'
- **getline** can stop reading at a character specified in the argument list
  - This code stops reading when a '?' is read
```
string line;
cout <<"Enter some input: \n";
getline(cin, line, '?');
```

### 2.5.3 Mixing cin >> and getline, ignor

- Recall **cin >> n** skips whitespace to find what it is to read then stops reading when whitespace is found
- **cin >>** leaves the **'\n'** character in the input stream

```cpp
// sample41.cpp
//
#include <iostream>
#include <string>

using namespace std;

int main(int argc, const char * argv[])
{
    string s1,s2;
    cout << "input:";
    cin >> s1;
    //cin.ignore(100,'\n');
    getline(cin,s2);
    cout << "s1=" << s1 << endl;
    cout << "s2=" << s2 << endl;
    return 0;
}
```

- **ignore** is a member of the **istream** class
- **ignore** can be used to read and discard all the characters, including **'\n'** that remain in a line
  - **ignore** takes two arguments
    - First, the maximum number of characters to discard
    - Second, the character that stops reading and discarding

## 2.6 String Processing

- The string class allows the same operations we used with C-strings ... and more
  - Characters in a string object can be accessed as if they are in an array
    - last_name[i] provides access to a single character as in an array
    - last_name.at(i) provides access to a single character as in an array
    - Index values are not checked for validity!

### 2.6.1 Member Function length

- The string class member function length returns the number of characters in the string object:
  - Example:

```cpp
int n = string_var.length( );
```

### 2.6.2 Comparison of strings

- Comparison operators work with string objects
  - Objects are compared using lexicographic order (Alphabetical ordering using the order of symbols in the ASCII character set.)
  - == returns **true** if two string objects contain the same characters in the same order
    - Remember **strcmp** for C-strings?
  - <, >, <=, >= can be used to compare string objects

### 2.6.3 string Objects to C-strings

- Recall the automatic conversion from C-string to string:

```
char a_c_string[] = "C-string";
string_variable = a_c_string;
```

- strings are not converted to C-strings
- Both of these statements are **illegal**:
  - `a_c_string = string_variable;`
  - `strcpy(a_c_string, string_variable);`

### 2.6.4 Converting strings to C-strings

- The string class member function **c_str** returns the C-string version of a string object
  - Example:

```
strcpy(a_c_string, string_variable.c_str( ) );
```

- This line is still **illegal**

```
a_c_string = string_variable.c_str( ) ;
```

  - Recall that operator = does not work with C-strings