Unit 12

C++ Strings
Character Arrays

• Recall that in C/C++ string constants (the text in between " ") are just character arrays
  – Each character consumes 1 element in the array
  – Ends with the null character (e.g. 0 decimal or '\0' ASCII)

• This approach of using an array of char's to store a string is referred to as a C-String

```c
int main()
{
    char str1[7] = "CS 102"
    /* Initializes the array to "CS 102"*/
    str1[5] = '3'; // now "CS 103"
    cout << str1 << endl;
    // prints "CS 103"
    cin >> str1; // get a new string from the user (suppose user types "hello"
    cout << str1;
}
```

Program Output:

```
CS 103
hello
```

Computer Memory

<table>
<thead>
<tr>
<th>Addr:</th>
<th>520</th>
<th>521</th>
<th>522</th>
<th>523</th>
<th>524</th>
<th>525</th>
<th>526</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index:</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>str1:</td>
<td>'C'</td>
<td>'S'</td>
<td>' '</td>
<td>'1'</td>
<td>'0'</td>
<td>'2'</td>
<td>'\0'</td>
</tr>
</tbody>
</table>
C++ Strings

- In C++, the library adds a new object type named **string (C++)** and provides an easier alternative to working with plain-old **character arrays (C-language)**

- **Do's and Don'ts**
  - **Do** #include <string>
  - **Don't** need to declare the size (i.e. [7]), just assign
  - **Do** still use it like an array by using [index] to get individual characters
  - **Do** still use cin/cout with strings
  - **Don't** worry about how many characters the user types when inputting to a C++ string

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

int main()
{
    char str1[7] = "CS 102";
    /* Initializes the array to "CS 102"*/
    string str2 = "CS 102";
    /* Initializes str2 to "CS 102"*/

    str1[5] = '3'; // now str1 = "CS 103"
    str2[5] = '4'; // now str2 = "CS 104"

    cout << str1 << endl; // prints "CS 103"
    cout << str2 << endl; // prints "CS 104"

    cin >> str1; // If the user types more than 6 chars..uh oh!
    cin >> str2; // str2 will adjust to hold whatever the user types
}
```
What Do Strings Do

- Strings simply abstract character arrays
- Behind the scenes strings are just creating and manipulating character arrays but giving you a simplified set of operators and functions
- Can concatenate (append) to a string with the `+` operator

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

int main()
{
    string str2 = "CS 102";
    // str2 stores 6 chars. = "CS 102"

    str2 = "Computer Science";
    // now str2 stores 16 characters

    // Can append using '+' or '+=' operator
    str2 = str2 + " is cool";
    // now str2 stores 24 characters
}
```
String Size

- Strings track how many characters they are storing
- Call the `<stringname>.size()` function to get the string's size
  - Returns the actual number of real characters (and does not count overhead like the null character)

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

int main()
{
    string str2 = "CS 102";
    cout << str2.size() << endl; // 6

    str2 = "Computer Science";
    cout << str2.size() << endl; // 16

    str2 = str2 + " is cool";
    cout << str2.size() << endl; // 24
}
```
String Comparison

- Comparison operators **do not work** with plain old character arrays (C-Strings)
- C++ strings **do perform** lexicographic (alphabetical/dictionary-order) comparison when comparison operators (<, >, ==, etc.) are applied
  - "a" < "z" ? __________
  - "a" > "aa" ? __________
  - "ab" < "ba" ? __________
  - "aab" < "aac" ? __________

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

int main()
{
    char str1[4] = "abc";
    string str2 = "abc";

    if( str1 == "abc" ) // doesn't work
    {
    ...
    }
    if( str2 == "abc" ) // works..true
    {
    ...
    }

    if( str1 < "aac" ) // doesn't work
    {
    ...
    }
    if( str2 < "aac" ) // works..false
    {
    ...
    }

    string str3 = "acb";
    if( str3 > str2 ) // works..true
    {
    ...
    }
}
```
Substrings

- **C++ strings** allow you to produce a new string from a *substring of a current string*

- Call either of the 2 versions:
  - `.substr(start_index)` or
  - `.substr(start_index, length)` function on the string
    - 1\(^{\text{st}}\) version generates substring from starting index location all the way to the end of the string
    - 2\(^{\text{nd}}\) version generates substring from the starting index and includes the next 'length' characters
    - Note: when a function has the same name but different options for parameters we say the function is **overloaded**

- **Returns a new string**
  - Even if length is 1 (i.e. if length is 1 you might think you just get a char, but you still get a string)

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

int main()
{
    string str1 = "CS102";
    string str2 = str1.substr(2); // str2 = "102"

    str1 = "Hello World";
    str2 = str1.substr(6,2); // str2 = "Wo"

    str2 = str1.substr(0,1); // str2 = "H"

    return 0;
}
```
SOLUTIONS
String Comparison

- Comparison operators **do not work** with plain old character arrays (C-Strings)
- C++ strings **do** perform lexicographic (alphabetical/dictionary-order) comparison when comparison operators (<, >, ==, etc.) are applied
  - "a" < "z" ? TRUE
  - "a" > "aa" ? FALSE
  - "ab" < "ba" ? TRUE
  - "aab" < "aac" ? TRUE

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

int main()
{
    char str1[4] = "abc";
    string str2 = "abc";

    if( str1 == "abc" ) // doesn't work
        {...}
    if( str2 == "abc" ) // works..true
        {...}

    if( str1 < "aac" ) // doesn't work
        {...}
    if( str2 < "aac" ) // works..false
        {...}

    string str3 = "acb";
    if( str3 > str2 ) // works..true
        {...}
}
```