Lecture 12

More on strings

Member functions

I'm back

- Was in the hospital Sat morning until Monday night
  - All is well.
  - Thought I had a heart attack, just inflammation of the heart sack.
    - Not a health issue. ☺
- I am now way behind on some things.
  - Will be caught up by Sunday morning.

Things to know

- First off, homework 3 is now due Tuesday the 8th at 5pm in the homework box
  - Or class on Monday, as you wish.
- Exam is Thursday Feb 10th 6:30-8:30pm Rooms will be announced on Monday.
- Don't forget P2 is due Monday at 11:59pm.

Characters

- Until Monday we only had dealt with two data types
  - Doubles and ints
  - Both were for representing numbers
- Another data type is "char" for character
  - Used to represent a single letter.
- Groups of characters are called strings

C++ and strings

- It turns out C++ has two different ways of representing strings
  - One is “C strings”
    - This is just an array of characters terminated by a special character (called `/0’)
    - This is the way the C programming language does strings.
  - One is C++ strings
    - A “smart” array of characters
    - Has a lot of functions and other wackiness associated with them.

More on the two

- C++ strings are much more powerful and handy, but various older bits of code use only C strings.
  - In general we will use C++ strings in this class.
    - But as time allows I will come back to C strings later. A C++ programmer needs to be okay with C strings.

C++ strings and you

- C++ strings have a large number of “function members” associated with them.
  - A member function is a function associated with a data type
  - You access them just like a “data member” of a struct.
  - So if “a” is a string
    - a.size() would be calling the fail() “member function” associated with C++ strings.

C++ member functions (1/4)

- length() or size()
  - Returns the number of characters in the string
- c_str()
  - Converts the C++ string into a C string
- insert()
  - Inserts a string into the current string, starting at the specified position.

```
string str12 = "0123";
str12.insert (1,"XYZ");
cout << str12 << endl;
// "0XYZ123"
```

C++ member functions (2/4)

- erase()
  - Delete a substring from the current string.

```
string str13 = "abcdefghi";
str12.erase (5,3);
cout << str12 << endl; // "abcdei"
```

- replace()
  - Delete a substring from the current string, and replace it with another string.

```
string str14 = "abcdefghi";
string str15 = "XYZ";
str14.replace (4,2,str15);
cout << str14 << endl; // "abcdXYZghi"
```

C++ member functions (3/4)

- find(), rfind()
  - Search for the **first** occurrence of the substring str in the current string, starting at position pos. If found, return the position of the first character. If not, return a special value (called string::npos). The member function rfind does the same thing, but returns the position of the **last** occurrence of the specified string.

C++ member functions (4/4)

- substr()
  - Returns a substring of the current string, starting at position pos and of length n:

```
string str18 = "abcdefghi"
string str19 = str18.substr (6,2);
cout << str19 << endl; // "gh"
```

Other things you can do with strings

- "="
  - Assign. Right side can be C or C++ string
- +
  - Concatenates.
- ==, !=, >, <, etc.
  - Compare strings. > and < use alphabetic order
- []
  - Treats the string as an array of characters.
  - Returns a character, not a string!

```
while (i< input.size())
{
    thischar = input[i];
    thischar = toupper(thischar);
    roman_numeral_index = roman_numerals.find(thischar);
    if (roman_numeral_index == roman_numerals.npos)
    {
        cerr << " Error " << thischar << " invalid" << endl;
    }
    new_roman_numeral_value = roman_values[roman_numeral_index];
    if (new_roman_numeral_value <= last_roman_numeral_value)
    {
        total = total + new_roman_numeral_value;
    }
    else // handle IV and IX
    {
        total= total + new_roman_numeral_value -
               2*last_roman_numeral_value;
    }
    last_roman_numeral_value = new_roman_numeral_value;
    i = i+1;
}
```

```
while (i< input.size())
{
    thischar = input[i];
    thischar = toupper(thischar);
    romanNumeralIndex = romanNumerals.find(thischar);
    if (romanNumeralIndex == romanNumerals.npos)
    {
        cerr << " Error " << thischar << " invalid" << endl;
    }
    newRomanNumeralValue = romanValues[romanNumeralIndex];
    if (newRomanNumeralValue <= lastRomanNumeralValue)
    {
        total = total + newRomanNumeralValue;
    }
    else // handle IV and IX
    {
        total= total + newRomanNumeralValue -
               2*lastRomanNumeralValue;
    }
    lastRomanNumeralValue = newRomanNumeralValue;
    i = i+1;
}
```