C++ string 정리 (C++ 문자열)

C++11 환경에서 정리한 글입니다
또한 `using namespace std`를 한 상태임을 밝힙니다
이 글을 통해 `std::string`을 간략하게 정리한다.

string 생성

방법1

`string myString = "abcd";`
단, 이 방식으로는 `'a'`와 같은 `char`로 생성이 불가능하다. 따라서 이 한계를 극복하려면 방법 2를 써야한다.

방법2

```
string myString;
myString = "abcd";
```

string 확장, 문자열 추가

방법1: += 연산자 이용

```
string base = "hello world!";
base += "x";
```

방법2: append() 멤버 함수 이용

```
string base = "hello world!";
base.append("appended!");
```

string 길이

```
string base = "hello world!";
base.length();
base.size();
```

> size()와 length()는 이름만 다를 뿐 같은 일을 하는 멤버 함수다.

메모리 관련

capacity()

```
string base = "hello world!";
base.capacity();
```

`capacity()`는 해당 문자열이 재할당을 하지 않고도 저장할 수 있는 문자열의 길이를 반환한다.
문자열은 문자열이 늘어났을 때, 현재 capacity보다 클 경우 더 큰 메모리를 사용할 수 있도록 재할당된다.

max_size()

```
string base = "hello world!";
base.max_size();
```

`myString.max_size()`는 최대한 메모리를 할당했을 경우, 저장할 수 있는 문자열의 길이를 반환한다.

string의 특정 위치 문자 받기(charAt)

```
string base = "hello world!";
base.at(0); // 'h'
base.at(1); // 'e'
```

해당 위치의 `char`를 반환한다.
java의 `String.charAt()`과 같다.

string에 있는 특정 문자 탐색

```
string base = "hello world!";
base.find("world!");
```

world! 문자열이 발견된 첫 위치를 반환한다.

```
if (base.find("world!") != string::npos) {
    // "world!"라는 문자열을 찾았을 때의 동작
}
```

탐색에 실패할 경우는 if 문에서 볼 수 있듯이 `string::npos`를 반환한다.

string간의 문자열 복사

```
string src = "I am source :)";
string dst;
dst = src;
```

dst에는 같은 내용이 복사되어 들어간다.
얕은 복사가 아니다. 깊은 복사다. 즉, 복사 후에 src의 내용이 변경된다고 해도 dst의 내용에는 아무 영향을 끼치지 않는다.

string간의 문자열 비교

```
string a = "I am string one! ;)";
string b = "string
if (a.compare(b) == 0) {
    // 두 문자열이 같을 때
} else if (a.compare(b) < 0) {
    // a가 b보다 사전순으로 앞일 때
} else if (a.compare(b) > 0) {
```

```
    // a가 b보다 사전순으로 뒤일 때
}
```

string의 문자열 대체하기 (replace기능)

http://stackoverflow.com/a/14678964/2050087 참조

Immutable Replace

원본 문자열에는 아무 영향을 끼치지 않는다. 변경된 문자열은 함수의 반환값으로 돌아온다.

```
#include <string>
#include <iostream>
using std::string;
std::string replaceString(std::string subject, const std::string &search, const std::string &replace) {
    size_t pos = 0;
    while ((pos = subject.find(search, pos)) != std::string::npos) {
        subject.replace(pos, search.length(), replace);
        pos += replace.length();
    }
    return subject;
}
```

Mutable Replace

원본 문자열을 수정한다. 속도가 우선일 경우 사용하자.

```
void ReplaceStringInPlace(std::string& subject, const std::string& search,
                          const std::string& replace) {
    size_t pos = 0;
    while ((pos = subject.find(search, pos)) != std::string::npos) {
        subject.replace(pos, search.length(), replace);
        pos += replace.length();
    }
}
```

타입 변환

문자를 다른 타입으로 변경해야 할 필요가 있는 경우는 흔하다. 그래서 C++11에 들어 표준 라이브러리에 기본적인 타입 변환 기능이 추가됐다.

> 개발환경이 C++11을 지원해야 한다.

```
// int ---> string
string s;
int i = 10;
s = std::to_string(i);
// string ---> int
string s = "123";
int i;
i = std::stoi(s);
```