문자열 클래스

1. 문자로 부터의 생성, 문자열 (char *)로 부터의 생성
2. 문자열 길이를 구하는 함수
3. 문자열 뒤에 다른 문자열 붙이기
4. 문자열 내에 포함되어 있는 문자열 구하기
5. 문자열이 같은지 비교
6. 문자열 크기 비교(사전 순)

일단 필요한거
문자열 데이터가 저장된 공간을 가리키는 포인터

문자열 데이터의 길이

assign

처음에 아주긴 문자열을 할당하였습니다. 예를 들어서 그문자열의 길이가 1000바이트라고 생각

다시 짧은 문자열을 지정할때는 OK len만 줄이고 동적배열은 그대로

문제는 긴문자열을 지정할때에는 메모리를 삭제하고 하는건데 만약 동적배열이 여유가 있다면 그대로

할당할 문자열의 크기를 미리 예약해 놓는 reserve 함수와

현재 문자열의 할당된 메모리 크기를 나타내는 capacity 함수를 만들수있다.

마지막으로 임의의 위치의 문자를 리턴하는 함수

add

if???

작은 크기의 문자열을 자주 집어 넣는 경우 큰 크기의 문자열을 한번에 insert하는 작업보다 작은 크기의 문자열을 여러번 insert하는 작업 보다 작은 크기의 문자열들을 여러번 insert하는 명령을 많이 수행

str의 길이가 길다면 자잘하게 문자를 넣고있게된다면 통그게 메모리를 미리 reserve 해놓는게 필요합니다.

insert작업에서의 잦은 할당/해제를 피하기 위해 미리 메모리를 할당해놓기와 메모리를 할당해 놓되, 많은 자원을 낭비하지 않는다

코드

```
if(memory_capcity * 2 > string_length + str.string_length)
{
	memory_capcity *= 2;
else
{
	memory_capcity = string_length + str.string_length;
}
```

즉 새로 할당해야할 메모리 크기(string_length + str.string_length) 가 현재의 memory_dcapacity 의 두배 이하라면 아예 memory_capcity의 두배의 달하는 크기를 할당해버리는 것입니다.

그리고 물론 insert되는 문자열의 크기가 엄청 커서 memory capcity의 두배를 뚜이ㅓ 넘어버린다면 그냥 예약을 생각하지 않고 필요한만큼 할당 해버려요

find??

insert와 erase 이외에 매우 빈번한 작업 ->> find

알고리즘에 따라 성능이 크게 좌지우지 하는 경우가 있다

문자열을 검색하는 알고리즘은 수없이 많지만 어떤상황에 대해서도 좋은 성능을 발휘하는 알고리즘은 없다

그렇기에 특별한 알고리즘을 사용하는 경우에는 그 클래스의 사용 목적이 명확해서 그 알고리즘이 좋은 성능을 발휘 할수 있는 경우에만 ㅅ하용하는 것이 보통이다

가장 간단한 방법으로 find 알고리즘 구현

find함수는 find_from에서 부터 시작해서 가장 첫번째 str의 위치를 리턴하게 됨

str 이 문자열에 포함되어 있지 않다면 -1리턴

compare

문자열간의 크기를 비교하는 compare함수

크기를 비교한다는 뜻은 사전식으로 배열해서 어떤 문자열이 더 뒤에 오는지 판단

이함수를 이용해서 문자열 전체를 정렬하는 함수

int compare(myString& cmpStr);

일단 함수의 원형은 위와같이 *this와 str을 비교하는 형태