STL - 6주 2일

Wulong

이번 시간은 **실습**입니다. 지금까지 공부한 내용을 복습할 수 있는 문제를 풀어보며 컨테이너 사용법을 익혀봅니다.

다운 받은 "STL Wiki.txt" 파일을 실습에 사용하겠습니다.
문제에 있는 시간은 거의 모든 학생이 풀 수 있으리라 예상하는 시간을 적은 것입니다.

먼저 답이 있는 문제를 소개하고 이후 실습 문제를 제시합니다.

자! 그럼 시작합니다.

다음은 답이 있는 문제이다.

[실습] "STL Wiki.txt" 파일을 읽어 화면에 출력하라. **white space도 출력**해야 한다. (20분)

```
#include <iostream>
#include <fstream>
#include <string>
#include "String.h"
#include "save.h"
using namespace std;


string file { "STL Wiki.txt"s };

int main( )
{
    // file 읽기

    ifstream in( file );
    if ( !in ) {
            cout << file << " 파일 열기 실패" << endl;
            return 0;
    }

    char ch;
    while ( in >> ch )
            cout << ch;

    save( "소스.cpp" );
}
```

시작 문제라 머리 깨우는 시간, 비주얼 스튜디오 여는 시간, 타이핑 시간 그 외 필요한 시간 모두 포함하여 20분입니다. 앞으로 헤더 파일은 생략하겠습니다. 헤더는 필요한 것들만 포함시켜야합니다. 컴파일에 문제없다고 아는 헤더파일 다 적어 놓지 마세요.

이 프로그램은 파일에서 한 글자를 읽어 그대로 화면에 출력합니다. 그런데 white space는 표시되지 않습니다. 왜냐하면 형식을 갖춘 입력처리 방식인 이 문장 in >> ch에서 white space는 알아서 건너뛰기 때문입니다. 어떻게 하면 될까요? 이거 우리 이미 알고 있는 겁니다. 1주 강의의 save.cpp에 답이 있네요. 약간만 자세히 설명하겠습니다.

```
int ch;
while ( ch = in.get( ) )
        cout.put( ch );
```

ifstream의 get 멤버로 파일에서 한 글자씩 읽어오면 됩니다. 그런데 in.get()의 return 타입은 int입니다. 이것은 파일의 끝(**EOF**: End of File)을 검사하기 위해서 그런 것입니다. 이렇게 고치면 분명 white space를 읽어 화면에 표시하지만 프로그램이 끝나지 않습니다. EOF를 검사하지 않았기 때문입니다. 이렇게 바꿔야 하겠습니다. 참고로 윈도우 콘솔에서는 Ctrl+Z로 EOF를 입력할 수 있습니다. 이전에 입력해본 기억이 날 겁니다.

```
int ch;
while ( (ch = in.get( )) != EOF )
        cout.put( ch );
```

문제가 해결되었습니다. 조금 더 나아가 보겠습니다.

[실습] "STL Wiki.txt" 파일을 읽어 화면에 출력하라. **소문자를 대문자로 바꿔 출력하라.**

(10분)

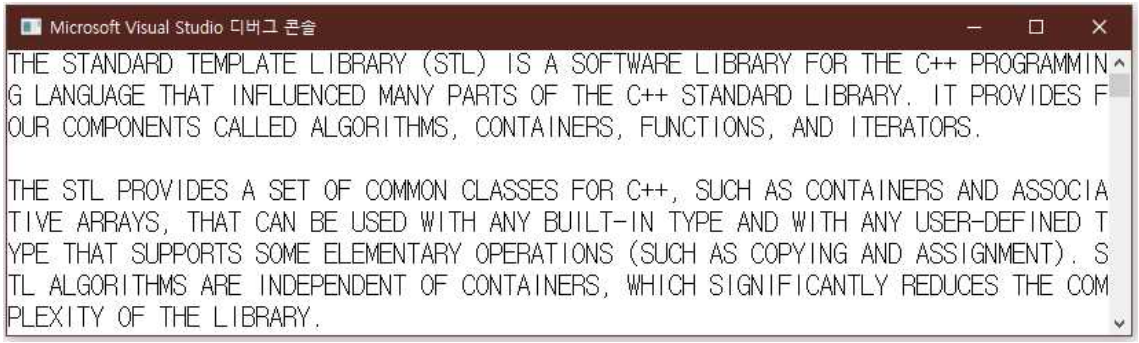
먼저 문제가 무엇인지 파악하고 어떻게 해결할지 생각합니다.
한 글자씩 읽을 수 있으니 읽은 글자를 그대로 출력하지 말고 검사를 해서 소문자라면 대문자로 바꿔 출력하면 되겠구나! 그렇습니다. 이렇게 생각할 수 있으면 바로 코딩하면 됩니다.

```
int ch;
while ( (ch = in.get( )) != EOF ) {

        // 화면에 출력하기 전 소문자라면 대문자로 바꾼다

        if ( isalpha( ch ) )                    // ch가 문자인가?
                if ( islower( ch ) )            // ch가 소문자인가?
                        ch -= 0x20;             // 그렇다면 대문자로 바꾼다
        cout.put( ch );
}
```

마지막은 조금 멋을 부려 봤습니다. 소문자에서 숫자 32를 빼면 바로 대문자이니까요? 그런데 이렇게 해주는 함수가 없겠습니까? 있습니다. 마지막 줄을 이렇게 바꾸면 모두 이해하기 쉽잖아요.

```
                                ch = toupper( ch );          // 그렇다면 대문자로 바꾼다
```

실행시켜 봅니다. 아주 잘 됩니다.



그런데 프로그램 좀 해 본 학생이라면 특히 C를 열심히 공부한 학생이라면, “아니 이걸 함수를 불러 해결한다고?” “함수호출에 따라오는 비용(보통 overhead라고 하죠)이 얼만데 이렇게 프로그램을 짜다니!”하며 혀를 찰지도 모르겠습니다.

그런데 컴파일러 만드는 사람들이 바보는 아닙니다. 이런 함수들은 정말 중요해서 최적화가 아주 잘되어 있거든요. 함수호출 안하고 사람이 직접 짠 코드처럼 호출하는 곳에 그냥 코드를 갖다가 놓습니다. 이런 걸 인라인(inline)이라고 합니다.

자! 그럼 위에 프로그램 만족하나요?

논리적으로 잘못된 곳이 없어 보이지만 이것은 컴파일러도 어떻게 해 볼 도리가 없는 프로그램입니다. 쓸데없는 중복이거든요. 이렇게 하면 충분합니다.

```
        int ch;
        while ( (ch = in.get( )) != EOF ) {
                ch = toupper( ch );                  // 대문자로 바꾼다
                cout.put( ch );
        }
```

toupper 함수가 알아서 소문자인 경우에만 대문자로 바꿔 줍니다.

내가 짠 코드가 실행된다고 프로그램이 아닙니다. 좋은 프로그램을 작성하려면 의심하고 의심해 보는 겁니다. 어디 더 손 볼 곳은 없는지. 무식해서 용감한 것은 아니지.

왜 정렬 알고리즘은 그렇게 여러 가지가 있는 것입니까? 정렬 알고리즘은 모두 원하는 대로 데이터를 정렬할 수 있습니다. 끊임없이 의심하고 개선하는 겁니다. STL의 sort도 초기 알고리즘과 지금 알고리즘은 다릅니다.

내가 강의에서 소개하고 설명하는 코드도 그냥 받아들이면 안됩니다. 나도 내가 어디서 잘못하고 있는지 모릅니다. 이건 순전히 무식해서 그런 겁니다. 다른 이유가 있을 수 없습니다. 그렇지만 **프로그램 세계에서는 잘못하는 것이 큰 문제가 되지는 않습니다.** 얼른 잘못했다고 인정한 후 반성하고 고치면 됩니다. 덧붙입니다. 어딘가에 프로그램을 내보내기 전까지 그렇다는 말입니다. 프로그램이 잘못 되서 큰 사고가 난 적은 셀 수 없이 많습니다.

[실습] "STL Wiki.txt" 파일의 소문자를 모두 대문자로 바꿔
"STL Wiki 대문자.txt"에 저장하라. (5분)

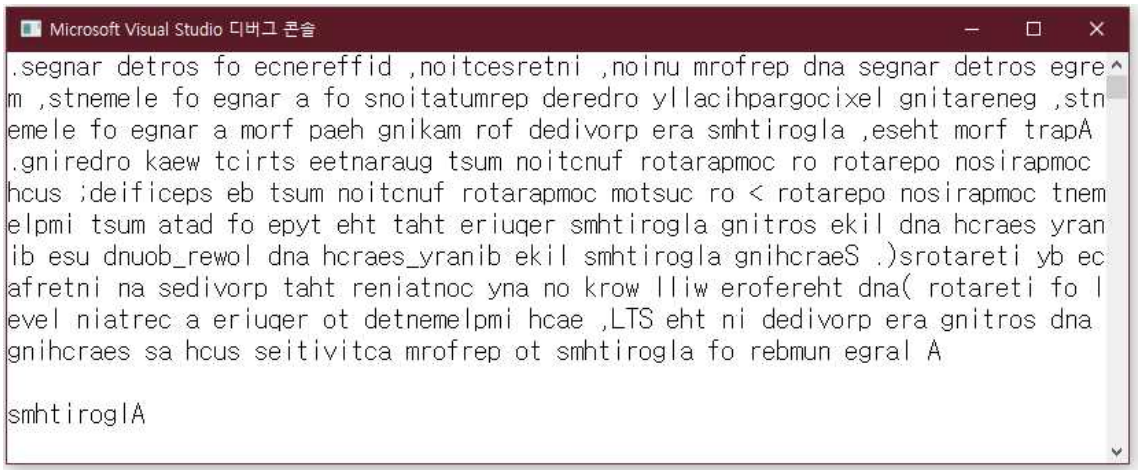
출력의 방향을 바꾸는 것은 일도 아닙니다. 얼른 몇 글자 타이핑해 보겠습니다.

```
string file { "STL Wiki.txt"s };
string out_file { "STL Wiki 대문자.txt"s };

int main( )
{
    // file 읽기

    ifstream in( file );
    if ( !in ) {
            cout << file << " 파일 열기 실패" << endl;
            return 0;
    }

    ofstream out( out_file );
    // if ( !out )으로 검사할 이유는 없습니다.
    // 이게 실패하면 컴퓨터 리셋하거나 하드 다시 사야합니다.

    int ch;
    while ( (ch = in.get( )) != EOF ) {
            ch = toupper( ch );         // 대문자로 바꾼다
            out.put( ch );              // out에 출력한다.
    }

    save( "소스.cpp" );
}
```

실행 결과는 파일을 열어서 확인하면 되겠습니다.

그럼 **지금부터 기술 들어갑니다.**
긴장하지 마세요. 답을 알려주는 문제입니다.

[실습] "STL Wiki.txt" 파일을 **거꾸로 출력**하라. 다음과 같이 출력되면 된다. (20분)



제일 뒤에 있는 글자를 제일 앞에 써야합니다. 뭐가 핵심입니까? 어떻게 하면 되죠? 핵심은 파일을 모두 읽은 후에야 이런 일을 할 수 있다는 것입니다. 물론 말이죠. 이 프로그램이 느리게 실행되도 문제없다면 파일포인터(FILE*)를 이용해서 이런 일을 할 수도 있습니다. 그렇지만 바보같은 일입니다. 더 편하고 쉽게 할 수 있는 방법이 있는데요.

핵심이 뭐라구요? 읽은 파일을 저장해야 한다는 것입니다.
"**저장**"하면 뭐가 떠 올라야합니까? STL 배운 후라면 바로 "**컨테이너**"가 떠올라야죠.
컨테이너하면 또 뭐가 먼저 떠오르죠? 그렇습니다. **vector**입니다.

vector에 읽어 저장하겠습니다.

```
string file { "STL Wiki.txt"s };

int main( )
{
    ifstream in( file );                    // 검사 생략

    vector<char> v;
    v.reserve( 2'000 );
```

```
        int ch;
        while ( (ch = in.get( )) != EOF ) {
                v.push_back( ch );
        }

        for ( char c : v )
                cout << c;
}
```

어떻습니까? 일단 닥치고 실행해 봅니다. 거꾸로 출력하라고 했지만 저장에 성공했으니 거의 다 한 기분이 듭니다. 여기서 vector<int>가 아니고 vector<**char**>에 저장하는 거 잘 보시고요.

한 2000글자 예약하고 시작하는데 이건 파일 크기가 얼마라는 사전 정보가 없어 그렇습니다. 그렇지만 나쁘지는 않아요. 우리 관찰해 봤잖아요. 메모리 크기를 2000에서 시작하면 다음에 3000, 그 다음에 4500으로 늘어납니다. 이런 식으로 늘어나니 재할당이 그렇게 빈번하게 일어나지 않을 거 같은 기분이 드니까 말이죠. 한 1'0000 글자 예약하고 시작해도 좋겠습니다. 10KB가 큰 메모리는 아니잖아요. 물론 더 좋은 방법도 있습니다. C++17의 filesystem을 이용하면 파일 사이즈가 몇 바이트인지 알 수 있거든요.

자. 그럼 거꾸로 출력은 어떻게 하면 좋을까요. 지금 프로그램의 출력은 컨테이너의 모든 원소를 range-based for로 출력하고 있습니다. 생각을 이렇게 해 보면 어떻겠습니까?

컨테이너의 원소를 처음부터 끝까지 훑어나가는 반복자가 있다.
그렇다면 끝부터 처음까지 훑어나가는 것이 그렇게 어렵지는 않을 것이다.
컨테이너가 **반대 방향으로 원소를 순회하는 반복자**를 주면 되겠구나!

그렇습니다. 양방향 이상의 반복자를 제공하는 컨테이너는 **역방향 반복자**(reverse iterator)를 제공합니다. 그렇다면 역방향 출력은 너무 간단합니다. 출력부분만 바꾸면 됩니다.

```
        for ( auto i = v.crbegin( ); i < v.crend( ); ++i )
                cout << *i;
```

먼저 실행시켜 출력을 감상하고 코드는 조금 뒤에 보겠습니다.

잘 감상하셨겠죠?

v.crbegin()은 읽기전용의 역방향 반복자를 리턴합니다. 그러니 auto i를 제대로 쓰면

```
vector<int>::const_reverse_iterator i = v.crbegin();
```

이렇게 됩니다. 그런데 재미있는 점이 무엇이냐 하면 제일 뒤에 있는 원소를 가리키는 역방향 반복자가 ++ 연산으로 v.crend() 까지 전진한다는 것입니다. 한 걸음씩 앞으로 가라는 연산을 계속하면 어떻게 제일 처음까지 가는 걸까요? 상황파악 되셨죠?

그렇습니다. 연산자 오버로딩때문입니다. ++는 operator++() 함수니까 이 함수 속에서 앞으로 가라는 동작대신 슬슬 뒷걸음치는 겁니다. 완전 사기죠. 그렇지만 이런 사기 동작 때문에 STL 사용자는 원소를 거꾸로 순회하는 동작을 정말 편하게 할 수 있는 것입니다. 똑바로 순회하는 코드에서 'r' 한글자만 덧붙이면 거꾸로 가니까요. 이렇게 사기 동작을 하는 반복자를 **반복자 어댑터**라고 하는 데 몇 종류가 더 있습니다. 다음 시간에 정리하겠습니다.

이제 기초를 잘 다진 것 같습니다. 안 배운 것이긴 하지만 stream에서 읽는 동작을 소개하고 답이 없는 실전 실습문제를 죽 나열해 보겠습니다.

STL에서는 반복자 어댑터인 **스트림반복자**를 사용하여 이렇게 파일을 읽을 수도 있습니다.

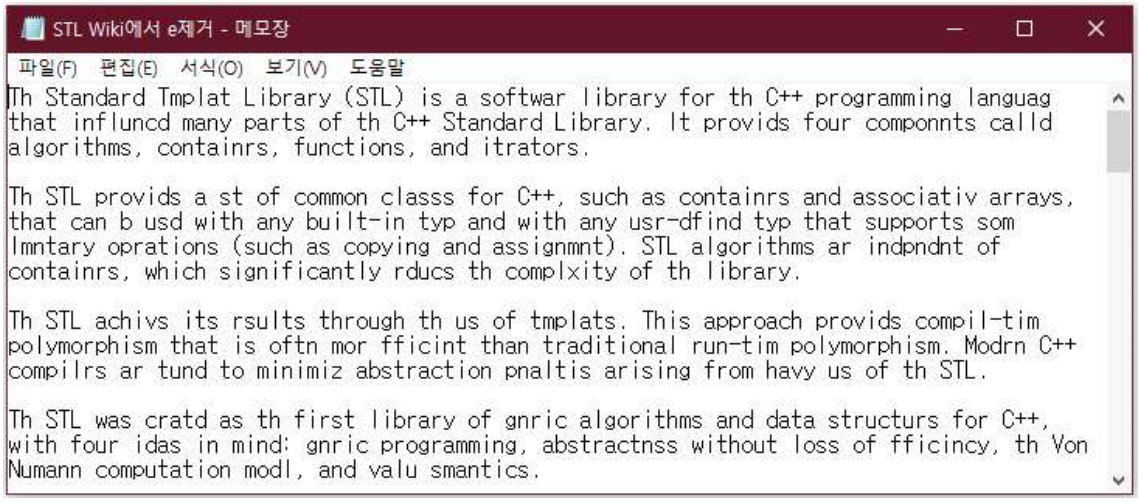
```
#include <iterator>

string file { "STL Wiki.txt"s };

int main( )
{
    ifstream in( file );                    // 검사 생략

    vector<char> v { istreambuf_iterator<char>( in ), istreambuf_iterator<char>( ) };

    // 모든 원소 출력
    for ( char c : v )
            cout << c;
}
```

white space를 제거하고 char만을 읽을 때는 이렇게 합니다.

```
vector<char> v { istream_iterator<char>( in ), istream_iterator<char>( ) };
```

실전 문제들입니다. 설명은 없습니다.
그렇지만 여러분은 문제를 해결할 수 있습니다.
코드를 여기 적진 않겠지만 **나도 문제를 풀겠습니다.**
그리고 **실행 결과 화면을 캡처**하여 덧붙이겠습니다.

[실습 1] "STL Wiki.txt" 파일에서
문자 **e를 제거**한 결과를 "STL Wiki에서 e제거.txt" 파일에 기록하라. (10분)



["STL Wiki.txt" 파일에서 e를 지운 파일을 메모장에서 열어 캡처]

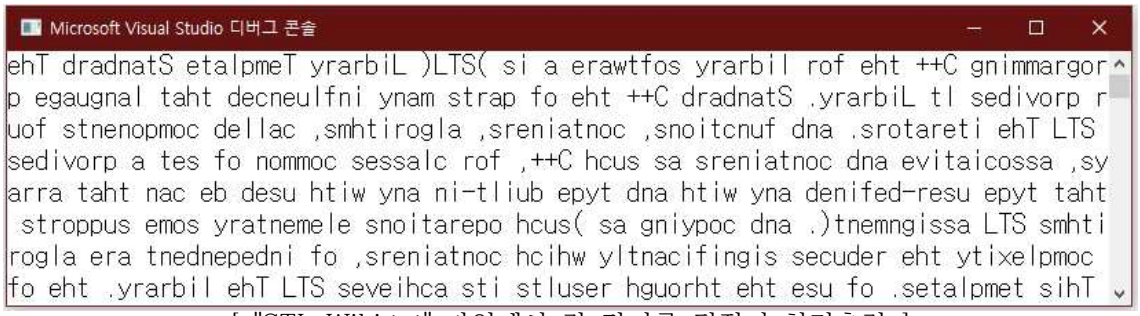
[실습 2] "STL Wiki.txt" 파일의 **단어 개수**를 출력하라. (10분)

이 실습 문제에서 단어의 의미 - 공백으로 구분되는 문자의 집합으로 정의함.

예) 위의 출력 창 3번째 줄에는 다음 5개의 단어가 있다.
algorithms, containrs, functions, and itrators.

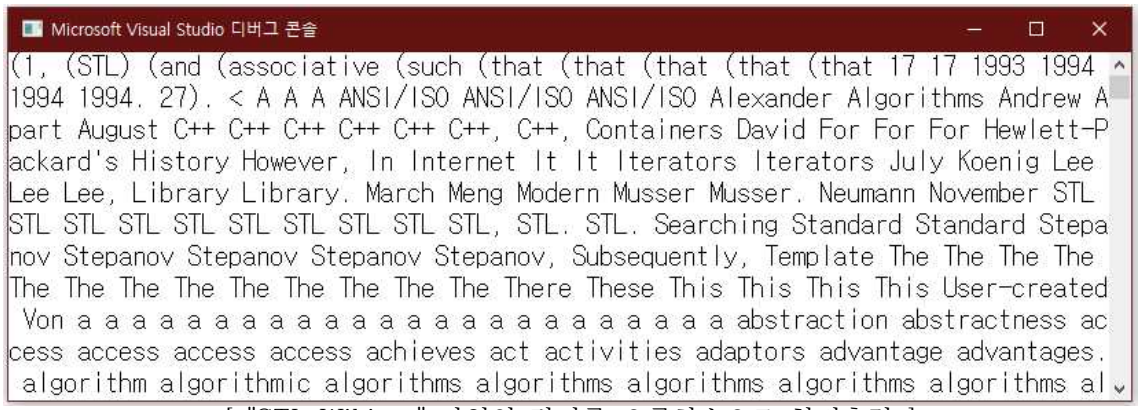
["STL Wiki.txt" 파일에 있는 단어의 수]

[실습 3] "STL Wiki.txt" 파일의 **단어를 뒤집어 출력**하라. (10분)



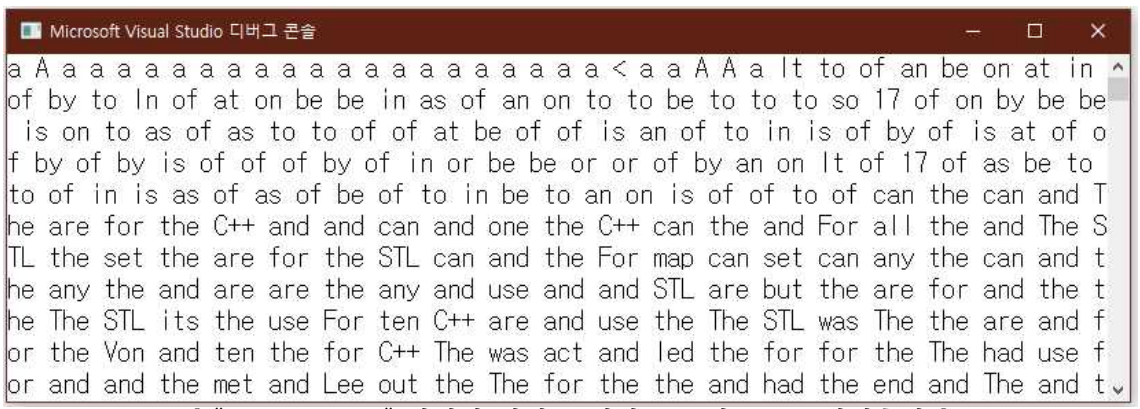
["STL Wiki.txt" 파일에서 각 단어를 뒤집어 화면출력]

[실습 4] "STL Wiki.txt" 파일의 **단어를 오름차순으로 출력**하라. (10분)



["STL Wiki.txt" 파일의 단어를 오름차순으로 화면출력]

[실습 5] "STL Wiki.txt" 파일의 단어를 **길이 오름차순으로 출력**하라. (10분)



["STL Wiki.txt" 파일의 단어를 길이 오름차순으로 화면출력]

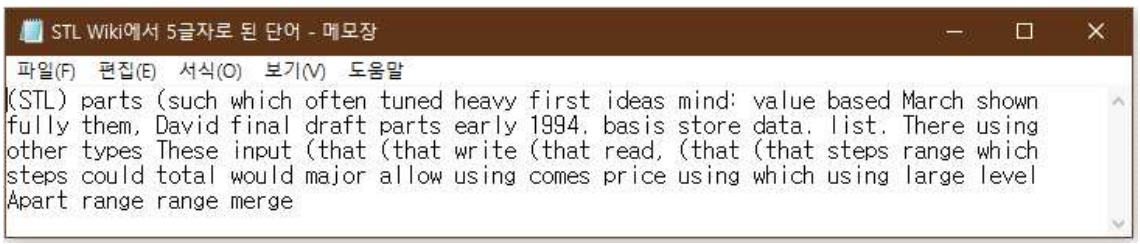
[실습 6] "STL Wiki.txt" 파일에서 **단어 the는 몇 개인지 출력**하라. (20분)

(설명) 모두 소문자인 the의 개수를 센다.
세다 - count.
조건에 맞는 것의 개수를 세다 - count_if

안 배운 함수라 구글링이 필요하지만 여러분은 처음 보는 알고리즘 함수를 사용할 수 있는 실력이 이미 있습니다.

["STL Wiki.txt" 파일에는 the가 34개 있다]

[실습 7] "STL Wiki.txt" 파일에서
다섯 글자인 단어를 따로 "STL Wiki에서 5글자로 된 단어.txt"에 저장하라. (10분)



["STL Wiki에서 5글자로 된 단어.txt" 파일을 메모장으로 열어 캡처]

[실습 8] 지금까지의 문제들을 사용한 **컨테이너를 바꿔 다시 풀어보라.** (30분)

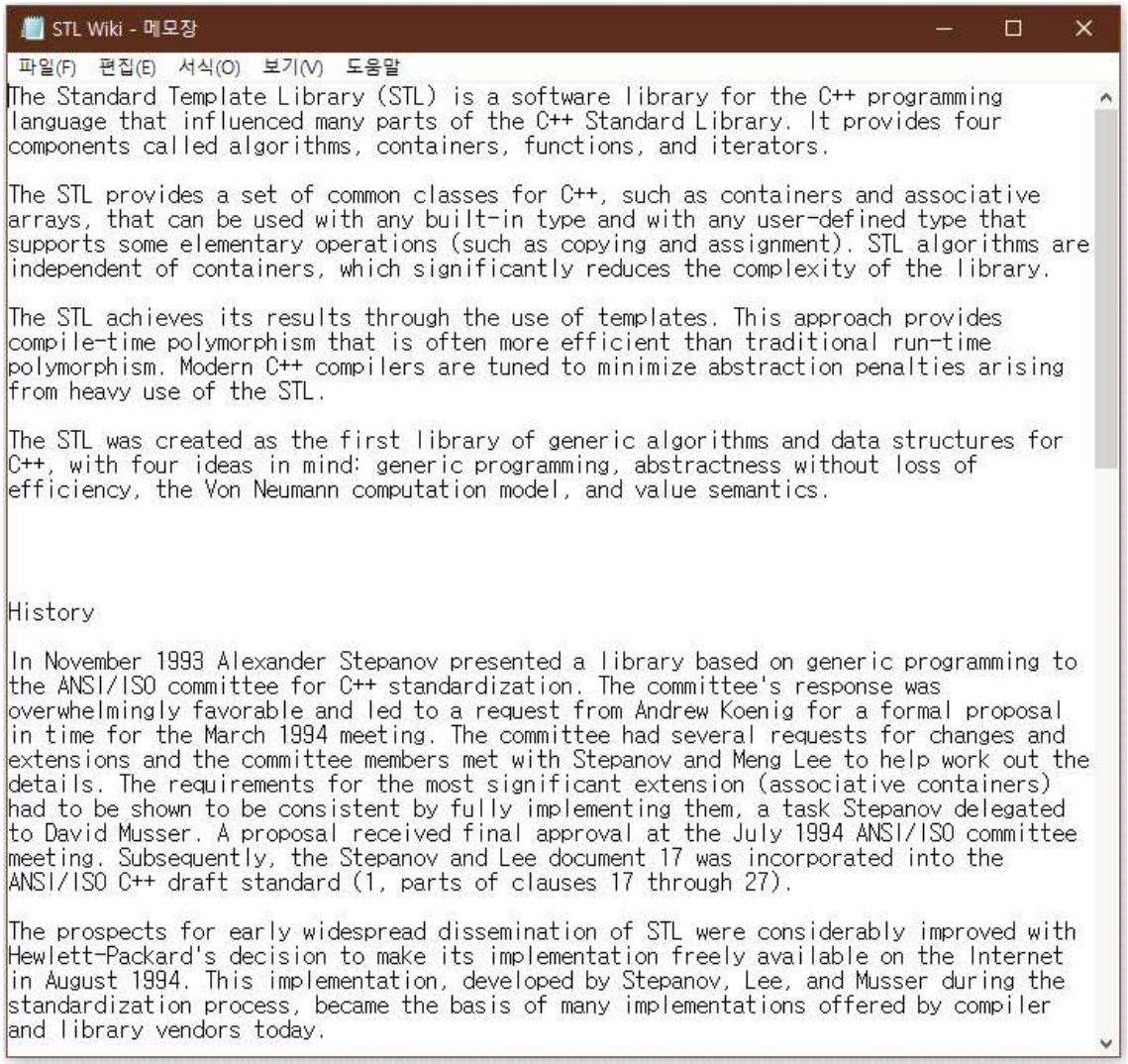
다들 vector로 문제 풀었죠? vector보다 더 빠르게 문제를 풀 수는 없나요?
array는 정보를 모르니 넘어갈 수밖에 없겠습니다만 지금 이 문제들은
vector, deque, list 어느 것을 사용하더라도 해결할 수 있습니다.

다 끝난 줄 알았죠? 이 실습 문제들 난 30분밖에 안 걸렸다. **이것도 문제라고 낸 거냐**. 너무 쉽다. 이런 학생들 분명 있습니다.

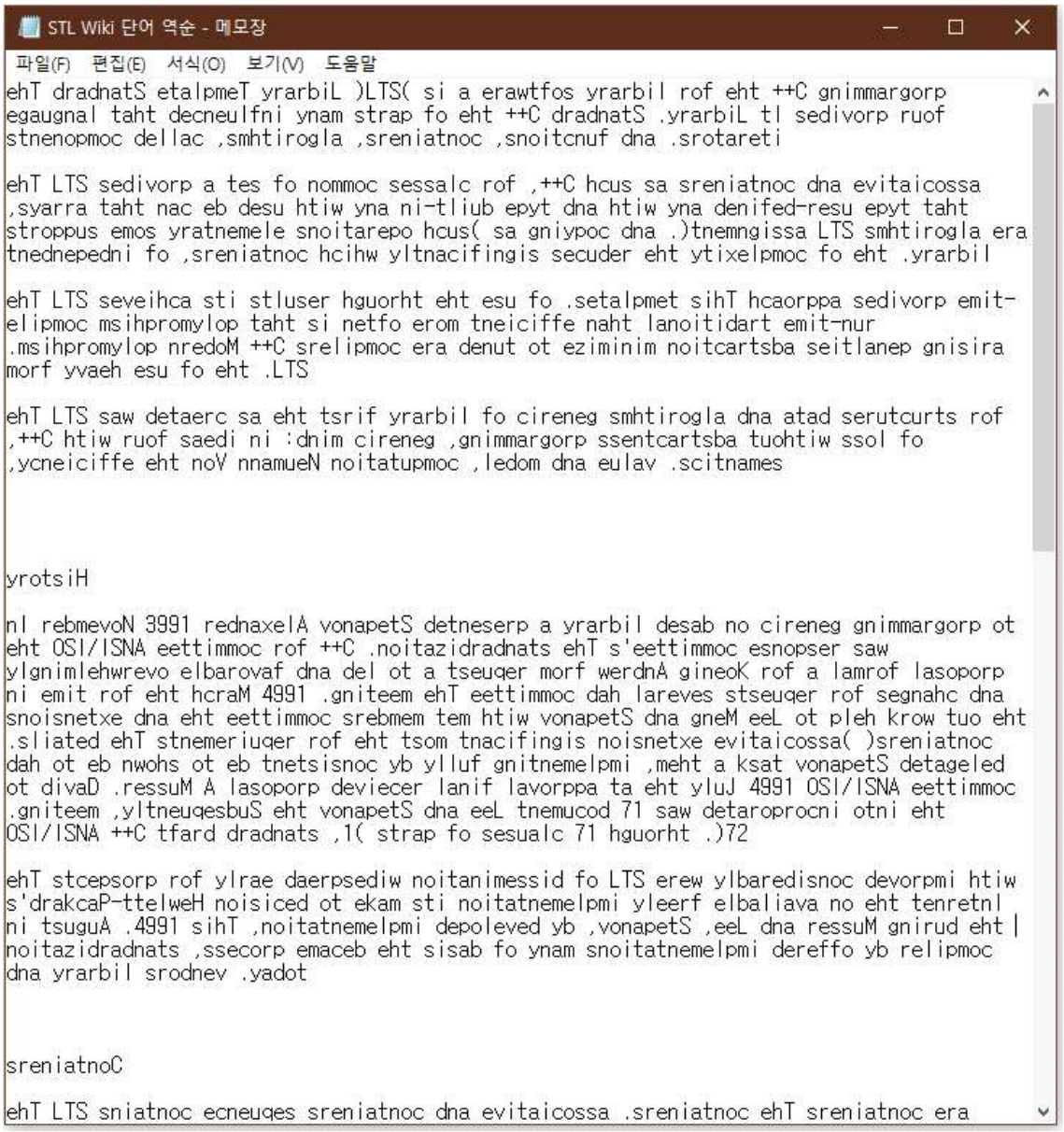
그래서! **도전 문제** 하나 나갑니다!

내가 나이 30대 까지만 해도 30분 안 걸렸을 거 같은 데 지금은 1시간으로는 턱도 없네요. 여러분은 아마 한 시간 안 걸려 해결할 수 있을지도 모를지도 아닐지도 모르겠습니다. ^^. 분명히 지금까지 강의 내용으로 이 문제를 해결할 수 있습니다.
결과 화면과 똑같이 나와야 됩니다.

[도전] "STL Wiki.txt" 파일에서
단어의 글자를 역순으로 바꿔 "STL Wiki 단어 역순.txt" 파일로 저장하라. **(1시간+)**



["STL Wiki.txt" 파일을 메모장으로 열어 캡처]



["STL Wiki 단어 역순.txt" 파일을 메모장으로 열어 캡처]

이 [도전] 문제를 해결한 학생은
얼마나 걸려 해결했는지
강의자료에 댓글 달아주세요!

[과제] 이 실습 문제들 모두 해 보세요.

언제나 그렇듯 모르는 내용은 다른 사람과 적극적으로 상의하는 것을 권장합니다.