STL - 4주 1일

Wulong

4주 강의 2020. 4. 6 ~ 2020. 4. 10	
지난 주 3.30 ~ 4.3	**객체지향 프로그래밍과 제네릭 프로그래밍** x86 프로세서의 메모리 구조 다수의 클래스 객체를 사용하는 프로그램 작성 호출가능 타입 스마트 포인터
이번 주 4.6 ~ 4.10	**STL 컨테이너** STL 클래스 준비 array vector
다음 주 4.13 ~ 4.17	**STL 컨테이너** vector의 메모리 관리 deque

지난 주

안녕하세요? STL 수강생 여러분!
2주 예정이었던 것이 언제까지 갈 지 알 수 없는 상황입니다.
이 자료가 알차다고 생각합니다만, 학생들을 강의실에서 졸지 못하게 괴롭히면서 한 줄 한 줄 코드를 쳐 가며 단계별로 프로그램이 변해가는 모습을 보이면서 강의하지 못하는 점이 아쉽습니다. 돌아보면 그 때 그 때 대답해 주는 학생들과 이야기하며, 상황에 맞춰 내가 아는 모든 것을 설명해 보려 노력하였지만 대부분의 경우에는 한 번에 컴파일을 성공하지 못하였습니다. 그럴 때 학생들에게 어디가 잘못됐는지 도와 달라고 하면 꼭 찾아주는 학생이 있어 다시 힘을 얻어 강의를 계속하곤 했었습니다. 학생들과 이야기 하며 코딩하는 과정을 보이는 게 내 강의의 핵심인데 지금은 그렇게 할 수 없어 아쉽습니다.

지난 주 강의자료는 재미있었습니까? 잘 공부했습니까? 사인 곡선이 잘 그려지던가요? 호출 가능 타입이 어떤 것인지 이해하셨죠? 그런데 제출한 과제를 살펴보니 여러분들이 책을 읽지는 않는다는 것을 알 수 있었습니다. 제출한 내용 중에 객체 100개를 정렬하는데 왜 sort(a, a+100)과 같이 써야 하는가를 묻는 경우가 많았거든요. 과제 해설 영상에서 알고리즘의 범위를 찾아보라고 이야기하기도 했지만 이건 바로 우리 책 1.4절 있는 내용이거든요. STL이 무엇인가 소개하는 부분을 책이나 구글에서 찾아보면 알 수 있는 내용이기도 하고요. 다시 말씀드리지만 공부는 필요한 내용을 스스로 찾아보는 것입니다. 물론 이 자료가 훌륭하기는 하지만 내가 여기에 정리한 내용을 읽고 알 수 있었다고 그만 덮고 넘어가면 안 됩니다. 남이 알려주는 건 공부가 아닙니다. 공부의 기본은 의심해보는 겁니다. 여기 있는 내용이 정말 맞는 말인가? 내가 모른다고 사기치는 것은 아닐까? 계속 의심해 보고 맞는 지 비교 검증해 보아야 하는 것이 공부입니다. 그런 경우가 거의 없기는 하지만 내가 무식해서 잘못된 정보를 알려주는 경우도 있었습니다.

지난 주 강의 내용이 2시간 안에 볼 수 있는 것들은 아니었을 것입니다. 프로그래밍 과목은 눈으로 읽는 것이 큰 도움이 되지 않습니다. **코딩은 머릿속을 프로그래밍 언어로 옮겨 적은 것입니다**. 어떤 순서로 머릿속이 돌아갔는지는 만든 그 사람 아니면 이해하기 어렵습니다. 시간이 지나면 만든 사람도 그때 어떻게 만들었는지 생각나지 않는 것이 프로그램입니다. 따라서 프로그램을 읽어서 이해할 수 있는 사람은 이미 유사한 프로그램을 해 본 경험자란 말이 되는 것이고요. 여러분 대부분은 이 STL이 처음일 겁니다. 눈으로 코드를 보지 마세요. 한 줄 한 줄 치고 실행시켜 보고 생각해 보는 겁니다. 나라면 어떻게 했을까를. 그렇게 하면서 공부하면 2시간이 모자랄 겁니다. 나도 2시간 안에 못할 내용입니다.

3주 2일차 과제는 웬만한 책이나 싸이트 찾아도 안 나오거나 찾았어도 금방 이해할 수 없는 내용이었을 것입니다. 우리 책에서는 이 책의 범위를 넘어가니 다루지 않겠다고 써 있네요. 그렇다고 진짜 어려운 내용은 아닙니다. 이전 과제들이 너무 쉽다는 의견들이 있어 조금 깊이 있는 내용을 제시해 보았습니다. 과제 마감시간이 아직 지나진 않았지만 지금까지 제출한 것을 보면 이 문제를 한 학생은 거의 없습니다. 해결 한 사람이 몇 명 없으면 해설하진 않겠습니다.

이제 STL을 공부하기 위한 기본 준비는 어느 정도 한 것 같습니다. 앞으로는 새로운 것들을 공부해 보겠습니다. 흥미를 위해 예제를 제시하고 관련된 내용을 설명해 보겠습니다. STL이 어떻게 돌아가는 것인지 이해하려면 메모리를 관찰해 보는 것이 꼭 필요합니다. 따라서 관찰 클래스를 만들어 놓고 시작하겠습니다. 관찰 클래스는 내가 만들어 파일로 제공하고 간단하게 설명하겠습니다.

STL

STL은 **자료구조**와 **알고리즘**을 제공한다고 했습니다. 그런데 이들은 각각 적어도 한 학기 강의를 통해 시간을 들여가며 공부해야만 뭘 하는지 구경해 볼 수 있는 내용이었을 것입니다. STL은 어떻게 자료구조와 알고리즘을 함께 제공할 수 있을까요?

STL은 데이터를 목적에 맞게 구성할 수 있도록 몇 가지 자료구조를 제공합니다. 분명 자료구조 시간에 공부해 봤던 것들이죠. STL은 자료구조를 컨테이너(container)라고 부릅니다. 컨테이너는 클래스 템플릿으로 array, vector, stack, queue, deque, list, forward_list, set, unordered_set, map, unordered_map이라는 이름의 헤더파일에 정의되어 있습니다.

STL은 컨테이너에 저장되어 있는 데이터에 적용할 수 있는 다양한 알고리즘을 제공하는데 특수한 몇몇 함수를 제외하고는 모두 algorithm 헤더 파일에 함수 템플릿으로 정의되어 있습니다.

컨테이너에 들어 있는 원소를 대상으로 알고리즘이 원하는 대로 원소를 검색하거나 제거하거나 정렬할 수 있습니다. 그런데 이렇게 하는 데 컨테이너와 알고리즘은 서로를 모르도록 설계되어 있습니다. 서로를 모른다는 말은 알고리즘이 특정 컨테이너에 대해서만 실행되는 것이 아니고 어떤 컨테이너인지는 모르지만 컨테이너의 원소에 접근하여 원하는 동작을 한다는 것을 말합니다. 이런 개념 어디서 들어봤던 거죠? 네. 바로 그렇습니다. **제네릭 프로그래밍**입니다. 사실 컨테이너도 마찬가지입니다. STL의 컨테이너인 list가 int만 저장하는 자료구조이겠습니까? 그렇지 않습니다. STL list는 어떤 자료형이라도 저장할 수 있습니다. 다만 어떤 자료형이 STL 컨테이너에 저장될 수 있으려면 반드시 지켜야할 조건들이 있을 뿐입니다. 이런 것들을 앞으로 배우게 될 것입니다.

컨테이너와 알고리즘이 직접 관련되지 않도록 이 둘을 서로 연결해주는 것이 바로 **반복자**(iterator)입니다. 반복자는 유명한 디자인 패턴입니다. 반복자가 있어 STL은 매우 유연한 라이브러리가 되었고 앞으로 나올 미래의 C++ 표준에서 새로운 컨테이너와 알고리즘이 추가되더라도 이전 코드들이 문제없이 실행될 수 있는 것입니다. 이것도 앞으로 배울 것입니다.

컨테이너와 알고리즘 그리고 이 둘을 연결하는 반복자가 STL을 떠받치는 세 가지 기둥입니다.

이 강의는 먼저 간단한 예제들로 STL을 구경하는 것에서 시작하여 컨테이너, 반복자, 알고리

즘을 만들어 보며 STL이 어떻게 구성되어 있는 지 그 속을 자세하게 알아볼 것입니다.

먼저 STL의 강력함을 바로 알아볼 수 있는 문제를 제시하겠습니다. 실습 문제를 읽어보고 어떻게 프로그램하면 좋을 지 생각해 보세요.

[실습] 키보드로 입력한 단어를 오름차순으로 정렬한 후 화면 출력하라. (5분)

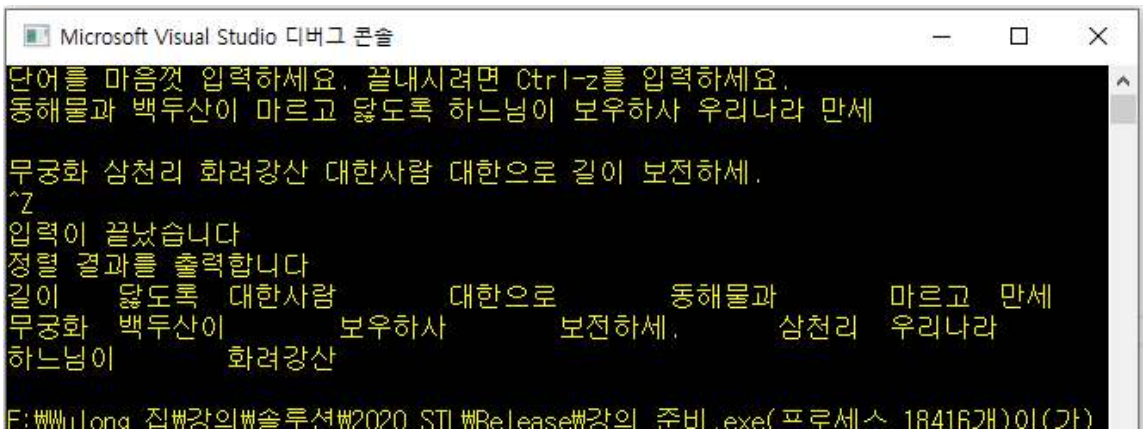
시간을 5분이라고 적어놓은 것은 문제를 풀라는 뜻이 아닙니다. 생각해 보라는 겁니다. 문제를 다시 천천히 읽어 보고 생각해 보세요. 문제를 해결할 수 없을 것입니다. 키보드에 손을 올려놓고 뭘 해 보려고 해도 막히는 곳이 있을 것입니다. 위의 실습 문장에서 어딘가 불편한 부분이 있습니까? 왜 불편한지 설명할 수 있습니까?

네. 이 문제는 도대체 몇 개 까지 입력하라는 것인지 정하지 않은 것이 문제입니다. 프로그램을 작성할 때 갯수가 정해지면 해볼 만한 문제인데 그렇지 않아 골치 아픈 것이죠. 해답은 프로그램이 실행될 때 입력되는 갯수에 맞게 메모리를 확보할 수 있는 자료구조가 필요하다는 것입니다. 그렇습니다. STL의 자료구조는 array 한 개를 제외하고는 모두 **동적(dynamic) 자료구조**입니다. 동적이란 말은 **"프로그램이 실행될 때"**라고 이해하면 되겠습니다. 반대말은 컴파일 시간(compile time)이라고 합니다. 그럼 코드를 한 번 적어 보겠습니다.

```
#include <iostream>
#include <vector>
#include <string>
#include <iterator>
#include <algorithm>
#include "save.h"

using namespace std;

int main()
{
    vector<string> v;

    string s;

    cout << "단어를 마음껏 입력하세요. 끝내시려면 Ctrl-z를 입력하세요." << endl;

    while ( cin >> s ) {
            v.push_back( s );
    }

    cout << "입력이 끝났습니다" << endl;

    sort( begin( v ), end( v ) );
```

```
    cout << "정렬 결과를 출력합니다" << endl;

    for ( auto&& s : v )
            cout << s << '\t';
    cout << endl;

    save( "소스.cpp" );
}
```

헤더파일이 여러 개 있습니다. 곧 익숙해 질 것입니다. 프로그램을 실행시킨 모습은 다음과 같습니다.



STL 코드이긴 하지만, 프로그램을 이해하기 쉽도록 최대한 늘여 쓴 것입니다. 실제로는 이렇게 STL 코딩을 하지는 않습니다. 대개 이렇게 합니다.

```
// 헤더 생략
using namespace std;

int main()
{
    cout << "단어를 마음껏 입력하세요. 끝내시려면 Ctrl-z를 입력하세요." << endl;

    vector<string> v { istream_iterator<string>( cin ), istream_iterator<string>( ) };
    sort( begin( v ), end( v ) );
    copy( begin( v ), end( v ), ostream_iterator<string>( cout, "\t" ) );

    save( "소스.cpp" );
}
```

어떻습니까? 이 코드에는 숫자가 없다는 것을 봐 주세요. 문제에도 숫자가 없었습니다. 동적

배열의 일종인 vector를 사용하여 문제를 해결하였습니다. 곧 살펴볼 STL의 대표 컨테이너입니다. 참 재미있겠죠? 다른 STL 컨테이너를 사용하면 위의 코드를 더 간단하게 작성할 수도 있습니다.

클래스 준비

STL을 제대로 감상해 볼 수 있게 준비한 관찰 클래스를 소개합니다. 클래스 이름은 String으로 하였습니다. 첫 글자가 대문자입니다. 사용자가 만든 클래스 이름은 대문자로 시작합니다. 소문자로 시작하는 클래스는 C++의 표준 class들입니다. 변수의 이름은 매우 중요한데요. 나는 멤버변수 작명에 두 가지만 지적하겠습니다.

- _로 시작하는 변수 이름을 사용하지 말자.
- m으로 시작하는 멤버 변수 이름을 사용하지 말자.

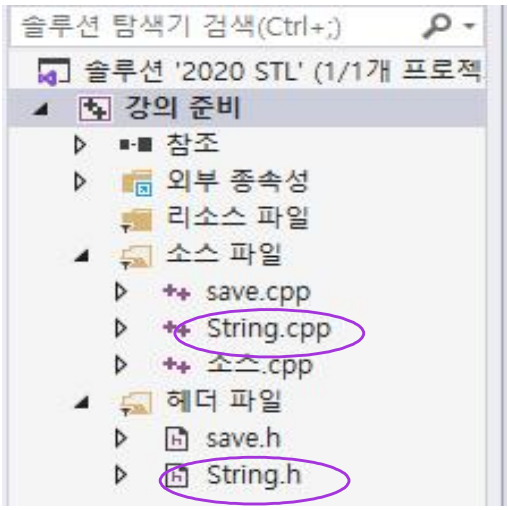
_로 시작하는 이름이 멋져 보이기는 합니다. int _num; 과 같이 쓰죠. 얼른 이런 습관 버려주세요. _로 시작하는 이름, 특히 __로 시작하는 이름은 사용자가 사용하지 않을 것이라 가정하고 비주얼 C++에서 심하게 사용합니다. 아무 헤더파일이나 열어서 잠깐 구경해 보세요. 또 하나 m이나 p등의 접두사를 붙여 작명하는 헝가리안 작명법을 사용하는 코드를 지금도 구경할 수가 있습니다. 어느 학원에서 가르치는지 궁금하기도 합니다만 학생들 코드에도 가끔 보입니다. 이런 코드를 보면 20년 전도 더 전에 사라진 코드를 아직도 사용하는 것이 신기하기도 하고 타임머신 타고 옛날로 돌아간 느낌입니다. 이름이 복잡한 다이렉트X 코드에서는 지금도 심하게 쓰고 있기는 합니다만 컴파일러가 똑똑해서 실시간으로 변수 이름을 구분할 수 있게 된 이후로 이렇게 공들여 작명하는 것은 사라졌습니다. 비주얼 스튜디오에는 인텔리센스가 열심히 이런 일을 합니다. 이런 코드를 쓰는 건 나는 구식이라고 일부러 광고하는 겁니다. 얼른 고쳐보세요.

String 클래스는 표준 string 클래스와 유사하도록 만들어 가겠습니다. 강의자료로 올린 파일을 프로젝트에 그대로 포함시키면 되겠습니다. 헤더 파일은 이렇습니다.

```
//------------------------------------------------------------------------
// String.h
//
// class String - STL 연습용 클래스
//  1. 컨테이너의 원소로 사용한다
//  2. char를 저장하는 STL 컨테이너가 되도록 코딩해 나아간다
//
// 2020. 4.  Programmed by Wulong
//------------------------------------------------------------------------
#pragma once

class String {
    size_t len { 0 };
```

```
    char* p { nullptr };

public:
    String( );
    String( const char* s );

    virtual ~String( );

    String( const String& other );
    String& operator=( const String& rhs );

    String( String&& other ) noexcept;

    String& operator=( String&& rhs ) noexcept;

    char& operator[]( size_t idx );
    char operator[]( size_t idx ) const;

    size_t size( ) const;
    std::string getString( ) const;

    friend std::ostream& operator<<( std::ostream& os, const String& s );
};
```

지금까지 3주간 공들여 복습한 것을 떠올리도록 주석을 붙이진 않았습니다. 언어 공부에 읽기도 필수입니다. 함수를 살펴보고 어떤 함수인지 얼른 대답할 수 있도록 익혀 두세요.



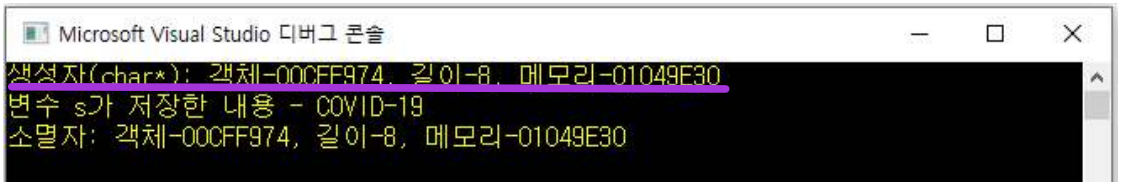
그림과 같이 솔루션 탐색기에서 프로젝트에 String 파일을 추가해 주세요.

[실습] String.cpp을 열어 코드를 읽어보라. (10분)

그럼 바로 연습해 보겠습니다.

```
#include <iostream>
#include "String.h"

using namespace std;

int main()
{
    String s { "COVID-19" };

    cout << "변수 s가 저장한 내용 - " << s << endl;
}
```

실행 시 출력화면 입니다.



String 객체 s는 지역객체이며 STACK에 생성됩니다. 생성과 소멸 시 관찰 메시지를 출력하였습니다. 객체 자체의 번지와 저장한 문자의 갯수 그리고 HEAP에 할당한 메모리 번지를 출력합니다. 관찰메시지를 출력하고 싶지 않으면 String.cpp 파일의 "#define 관찰" 문장을 주석처리하면 되겠습니다. 직접 출력하여 무슨 일이 일어나는지 관찰할 수 있다는 것이 내가 만든 객체의 좋은 점입니다. POD 타입은 이렇게 할 수 없습니다.

메모리가 살고 있는 곳은 3군데라고 했습니다. 아래 코드를 살펴보고 출력하여 결과를 보세요. 분명히 다른 곳에 살고 있죠? 내가 방금 생각한 코드지만 아름답습니다. 한 번에 다 관찰할 수 있다니요!

```
#include <iostream>
#include "String.h"
#include "save.h"

using namespace std;

String g { "COVID-19" };              // g는 DATA, "COVID-19"는 HEAP에 복사

int main()
{
    String s { "COVID-19" };          // s는 STACK, "COVID-19"는 HEAP에 복사

    int n { 2020 };
```

```
    cout << "지역변수 n의 번지 - " << &n << endl;          // n은 STACK

    save( "소스.cpp" );
}
```

아이디어가 계속 떠오르네요! 하나 더 보고 가죠.

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

int main()
{
    auto s = new String { "COVID-19" };

    cout << "변수 s가 저장한 내용 - " << *s << endl;
}
```

이런! 실행시켜 보니 뭘 잘못했는지 알겠네요. 제대로 해 보겠습니다.

```
#include <iostream>
#include <memory>
#include "String.h"
using namespace std;

int main()
{
    auto s = unique_ptr<String>( new String("COVID-19") );

    cout << "변수 s가 저장한 내용 - " << *s << endl;
}
```

이제 문제 없게 되었습니다. 위의 코드보다 더 바람직한 코드는 아래와 같이 쓰는 겁니다.

```
int main()
{
    auto s = make_unique<String>( "COVID-19" );

    cout << "변수 s가 저장한 내용 - " << *s << endl;
}
```

진짜 더 좋은지 확인해 볼 수 있으며 현대 컴파일러가 얼마나 코드를 잘 만들어내는지도 구경해 볼 수 있습니다. 순전히 관찰 메시지 덕분이지요.

그럼

이제

진짜

정말로

STL 컨테이너를 사용하러 가겠습니다. GO! Go!

array

제일 처음 만나는 STL 컨테이너입니다. **지금까지 공부해 놓은 것이 많아 어려움 없이 알아볼 수 있습니다.** array는 다른 STL 컨테이너와 같이 동일한 자료형을 담을 수 있는 자료구조입니다. 이름에서 상상해 본다면 이것은 **똘똘한 배열**([ ])입니다. POD 배열의 기능을 그대로 하면서 다른 일도 할 수 있도록 클래스로 만든 배열입니다. 그러면서도 **POD 배열과 사용하는 메모리의 크기는 같습니다**. 한 걸음 더 나아가 어떠한 자료형이라도 담을 수 있도록 템플릿으로 작성한 클래스입니다. array의 정의는 array 헤더에 있으며 언제라도 파일을 열어 어떻게 작성되어 있는 지 살펴볼 수 있습니다. 지금 살펴보니 500줄이 안되는 코드입니다.

그런데 array는 다른 모든 STL 컨테이너와는 다른 점이 있습니다. 다른 컨테이너는 저장할 원소의 갯수가 얼마인지 모르는 상태로 사용할 수 있지만 array는 그럴 수 없습니다. array는 자료를 저장할 공간을 동적으로 할당하지 않거든요. 2주 2일차 과제에서 살펴본 바와 같습니다. 진짜 그런지 살펴보겠습니다.

앗! 잠깐만요.
지금까지와 같이 구구절절한 설명은 안 합니다. 이 강의자료는 책이 아닙니다. 컨테이너 소개와 공통 멤버 함수 그리고 array<T,N> 사용하기는 책 2.1절과 2.2절을 봐 주세요.

```
#include <iostream>
#include <array>
using namespace std;

int main()
{
    array<int, 5> a;
    array<int, 10> b;

    cout << "a의 크기 - " << sizeof( a ) << endl;
    cout << "b의 크기 - " << sizeof( b ) << endl;

    cout << "a의 데이터가 저장된 메모리 - " << a.data() << endl;
```

```
    cout << "b의 데이터가 저장된 메모리 - " << b.data( ) << endl;

    int n { 2020 };
    cout << "STACK 비교를 위한 변수 n의 번지 - " << &n << endl;
}
```

실행화면을 보면 모든 메모리가 STACK에 있음을 알 수 있습니다. 벌써 a와 b의 메모리 크기가 다르다는 것이 동적할당을 하지 않는 다는 말이거든요.

array는 갯수가 고정되어 있고 앞으로도 변동될 일이 없는 경우에 사용하며 전체 메모리 크기도 STACK에서 사용하기에 적당하여야 합니다. 그러니 다음과 같이 많은 갯수를 담으려고 하면 안 됩니다.

```
int main()
{
    array<int, 100'0000> a;

    cout << "a의 크기 - " << sizeof( a ) << endl;
}
```

array는 POD 배열 보다 기능이 좋은 배열이라 생각하고 사용하는 겁니다. 당연히 경계 검사를 할 수도 있습니다. 그 밖에 멤버 함수들은 객체에 .을 타이핑해 보면 볼 수 있겠습니다. 지금까지 예는 진짜 프로그램이 당연히 아닙니다. array<int, 10>과 같은 프로그램을 짤 일은 전혀 없습니다. array<String, 3>과 같은 프로그램을 해야죠.

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

int main()
{
    array<String, 5> words { "corona", "virus", "world", "crisis", "pandemic" };
```

```
    cout << words[0] << endl;           // 배열과 같은 사용법

    auto p = words.begin( );            // words의 첫 원소를 가리키는 반복자
    cout << *p << endl;                 // 반복자 p가 가리키는 것은 바로 첫번째 원소

    ++p;                                // 다음 원소로 이동
    cout << *p << endl;

    for ( int i = 0; i < words.size( ); ++i )       // 모든 컨테이너는 size 멤버를 제공한다
        cout << words[i] << " ";
    cout << endl;

    for ( auto p = words.begin( ); p != words.end( ); ++p )         // 반복자로 원소를 순회
        cout << *p << " ";
    cout << endl;

    for ( const String& word : words )          // range-based for로 순회
        cout << word << " ";
    cout << endl;


    // 범위를 벗어난 경우를 굳이 점검하고 싶다면
    try {
        // 이렇게 쓰면 안되고
        // cout << "이건 없는 원소잖아?" << words[-1] << endl;

        cout << "이건 없는 원소잖아?" << words.at(-1) << endl;
        // 예외 때문에 endl이 출력되지 않을걸!
    }
    catch ( const exception& e ) {
        cout << "C++에서 예외를 사용하다니? - " << e.what( ) << endl;
    }

    save( "소스.cpp" );
}
```

[실습] 코드를 타이핑하고 결과를 관찰하자. (10)

이런 시간이 다 되었네요. 오늘 강의는 여기까지 입니다. 다음 시간에 만나요!

[4주 1일 - 과제]

[문제 1] 객체 words와 words 원소인 5개 String의 메모리를 그림으로 그려라.

이 과제는 자율과제입니다.
그림으로 그려보고 다른 사람과 맞는지 이야기 해 보세요.
문서로 제출할 필요 없습니다.