STL - 2주 1일

Wulong

2주 강의 2020. 3. 23 ~ 2020. 3. 27	
지난 주 3.16 ~ 3.20	**강의 안내** 컴파일 환경 강의를 저장하기 위한 save 함수 작성 파일 입출력 클래스 복습
이번 주 3.23 ~ 3.27	**템플릿과 제네릭 프로그래밍** 자료를 파일로 저장 파일에서 자료 읽기 자료 정렬 실행 시간 측정 템플릿과 제네릭 프로그래밍 array 템플릿 작성
다음 주 3.30 ~ 4.3	**객체지향 프로그래밍과 제네릭 프로그래밍** 다수의 클래스 객체를 사용하는 프로그램 작성 호출가능 타입 스마트 포인터

지난 주

안녕하세요? STL 수강생 여러 분!
지난 주 강의 자료로 재미있게 공부하셨죠?
잊었던 내용들이 잘 떠오르던가요? C++ 머리로 바꾸는데 시간을 좀 들여야 했죠?

여러분이 제출한 과제는 이번 주 동안 살펴보고 내 의견을 적어보겠습니다.

save 함수를 save.h와 save.cpp로 분리하였나요?
클래스 복습 참고자료에 있는 모든 문제를 해결할 수 있었습니까?

문제의 내용을 이해하여 빠른 시간에 원하는 자료를 찾아 해결할 수 있었다면 칭찬합니다.
꾸준히 C++ 언어를 공부하지 않았다면 답을 적기 어려웠을 거거든요.

클래스 복습 내용에 적혀있는 것들이 들어는 본 것 같은데 잘 기억나지 않는 학생들이 많았을 것입니다. 이전에 공부했던 기억이 있다면 빠른 시간에 복습이 가능할 것입니다. 주변에 좋은 사람들이 있다면 도움을 청해보세요. 서로의 실력을 높일 수 있을 것입니다.

자료에 적힌 것들이 무슨 말인지 모르는 학생들도 분명히 있을 것입니다. 강의를 듣지 말라고 하지는 않습니다. 용감하게도 C++ 언어를 공부한 적도 없고 자료구조와 알고리즘이 무엇인지 모르는 상태에서 수강하는 학생들도 꾸준하게 있었습니다. 그런 학생들은 아마 이번 강의에서 학점을 얻을 수 없을 것입니다. 그렇지만 다음에 다시 수강하거나 필요해서 스스로 공부한다면 지금 보다는 훨씬 나을 것입니다. 공부는 반복이니까요.

STL은 자료구조와 알고리즘을 제공합니다.
자료구조는 많은 자료를 읽고 쓰는 내용을 다룹니다.
알고리즘은 자료구조에 있는 자료를 어떻게 처리할까에 집중합니다.

처음 몇 주간 강의내용은 여러분에게 많은 자료를 처리하는 방법을 설명합니다.
언어가 제공하는 자료형을 어떻게 이용하는지 설명하는 것에서 시작합니다.
이후에 사용자가 만든 자료형을 STL에서 어떻게 다루는지 집중합니다.
이것이 이번 학기에 공부할 내용입니다. 아직은 조금 더 기초를 다져야겠습니다.

자료를 파일에 저장

지난 시간 강의에서 int를 파일에 저장하는 문제를 냈습니다. 자료를 파일로 저장할 때 어떻게 저장하는 것이 좋겠습니까. 만일 여러분이 int형 자료 1000개를 프로그램에서 사용하려면 sizeof(int) * 1'000 바이트만큼의 저장 공간을 정의할 것입니다. 아마도 int [1'000]과 같이

배열을 이용할 것입니다. sizeof(int)가 4 바이트라면 4'000 바이트의 공간이 있으면 저장할 수 있습니다. 그런데 문제에서 살펴 본 파일은 이것 보다 큰 크기였었죠? 왜 그랬던 건가요?

그것은 int를 저장하는데 int의 값을 사람이 알아볼 수 있는 숫자의 형태로 바꿔 저장하였기 때문입니다. int 값이 4자리수 보다 더 크면 실제 보다 더 많은 공간을 차지하게 되는 것이죠. 또 각 int를 구분하기 위한 공백도 필요합니다. 여기에서 생각해 볼 점은 어떤 자료를 저장할 때 사람이 읽어볼 수 있는 형태로 저장하는 것이 좋겠는가 그렇지 않은가입니다.

사람이 읽을 수 있는 형태로 저장하려면 메모리에 저장되어 있는 숫자를 아스키코드로 바꿔 저장합니다. 더 많은 메모리가 필요합니다.

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

int main()
{
    int num = 0B0111'1111'1111'1111'1111'1111'1111'1111;          // 가장 큰 양의 정수

    cout << "num의 메모리 크기는 " << sizeof( num ) << "바이트" << endl;
    cout << "num에 저장된 값은 " << num << endl;

    string s{ to_string( num ) };
    cout << "num을 글자로 저장하려면 " << size( s ) << "바이트가 필요합니다" << endl;
}
```

코드를 실행시켜 보세요. 확실하게 이해할 수 있습니다. 앞에 붙인 0B는 이진수를 의미하며 숫자 사이에 '를 사용하여 긴 숫자를 쉽게 구분할 수 있습니다. 32bit를 사용한 int 자료형에서 가장 큰 양의 정수는 2147483647이며 모두 10글자입니다. 이 값은 메모리 4바이트를 차지하는데 그 값은 코드 첫 줄에 있는 것과 같습니다. 16진수로 0x7FFFFFFF입니다.

사람이 읽을 필요가 없다면 메모리의 bit값을 그대로 저장하는 것이 가장 좋습니다. 대부분의 자료가 이렇게 저장됩니다. 음악파일이나 영화 또는 게임 데이터들을 우리가 읽을 수는 없습니다. 우리가 읽을 수 있도록 저장한다면 마음대로 값을 바꿀 수도 있을 것입니다.

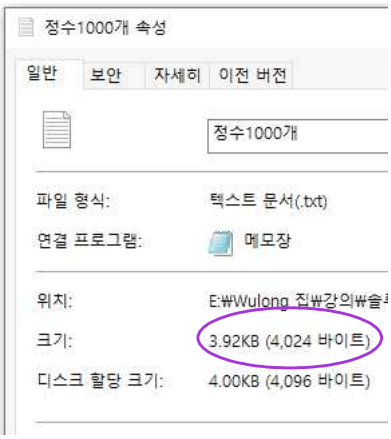
메모리의 값을 그대로 저장하려면 다음과 같이 코딩합니다. 특히 물리적으로 연속되어 있는 메모리를 읽거나 쓴다면 이것보다 더 빠른 방법은 없습니다. 찾아보고 연습하고 익숙해져야 합니다. 자료의 코드를 붙여 넣지 마세요. 생각해 보며 직접 키보드로 입력해야 합니다.

```
#include <iostream>
#include <random>
#include <fstream>
```

```
using namespace std;

int main() {
    default_random_engine dre;
    uniform_int_distribution uid;
    ofstream out( "정수1000개.txt" );

    int n;

    for ( int i = 0; i < 1'000; ++i ) {
            n = uid( dre );
            out.write( (char*)&n, sizeof( n ) );
            // out.write( reinterpret_cast<const char*>(&n), sizeof( n ) );
    }
}
```

ofstream의 멤버함수 write를 사용하여 저장합니다. 첫째 인자는 저장할 메모리의 시작번지이고 둘째 인자는 메모리의 바이트 수 입니다. 어떤 메모리라도 시작번지와 저장할 바이트 수를 지정하면 그대로 파일에 저장합니다. 가장 빠른 방법입니다.

폴더를 열어 파일 크기를 확인해 보세요. 확실하게 4000바이트의 하드디스크 공간을 차지하고 있는 지 말이죠. 파일이름을 마우스 오른쪽 버튼으로 클릭하면 컨텍스트 메뉴가 뜨는데 제일 아래 항목인 속성을 선택해 보세요.



정확하게 4000바이트입니까? 그렇지 않네요. 4024바이트입니다. 메모리 4000바이트를 저장하랬더니 24바이트를 더 사용했네요. 정수 1000개를 한 번에 저장하지 않고 한 개를 1000번 저장해서 이런 일이 생긴 걸까요? 왜 그런 걸까요?

텍스트 모드와 바이너리 모드가 무엇인지 검색할 때가 되었습니다. OS에 따라 파일을 텍스트 모드로 저장할 때 특별한 동작을 하기 때문입니다. 얼른 검색해 보세요. 5분 쉬겠습니다.

금방 알 수 있었죠? 그럼 메모리 그대로 저장하려면 어떻게 해야 된다는 것이죠?

그렇습니다. 컴퓨터에게
 - 이 파일은 읽을 필요가 없는 파일이다.
 - 바꿈 문자를 나중에 읽기 편하도록 다른 문자를 덧붙여 저장할 필요가 없다.
라고 알려주면 됩니다. 코드에서는 파일을 바이너리 모드로 열면 됩니다.

이제 프로그램을 수정하고 다시 실행해서 확인해 보겠습니다.

```
#include <iostream>
#include <random>
#include <fstream>
using namespace std;

int main() {
    default_random_engine dre;
    uniform_int_distribution uid;
    ofstream out( "정수1000개.txt", ios::binary );

    int n;

    for (int i = 0; i < 1'000; ++i) {
            n = uid( dre );
            out.write( (char*)&n, sizeof( n ) );
    }
}
```

파일에서 자료를 읽기

그럼 바이너리로 저장된 자료를 읽어 오는 것도 아무 문제없이 할 수 있겠죠?
"정수1000개.txt" 파일에 저장된 정수 중 가장 큰 정수를 찾아 화면에 출력하는 프로그램을 작성해 보겠습니다.

여러분도 내 방식을 보기 전에 한 번 코딩해 보세요.

```
#include <iostream>
#include <fstream>
#include <limits>
using namespace std;
```

```
int main() {
    ifstream in( "정수1000개.txt", ios::binary );

    int n;
    int max { numeric_limits<int>::min() };                    // 가장 작은 값으로 설정

    for (int i = 0; i < 1'000; ++i) {
            in.read( (char*)&n, sizeof( n ) );
            if (max < n)
                    max = n;
    }

    cout << "\"정수1000개.txt\"에 저장된 int 중 가장 큰 값은 " << max << endl;
}
```

각각 한 학기 동안 배웠던 자료구조와 알고리즘을 제공하는 **STL을 제대로 활용하기 위한 준비 작업을 하는 중**입니다. 지금은 기본자료형(POD: Plain Old Data)으로 많은 자료를 다뤄보고 있습니다만 앞으로는 class 객체를 다루게 될 것입니다. 템플릿까지 복습한 후 본격적으로 STL을 공부하겠습니다.

정렬

이제 많은 자료를 읽고 쓸 수 있게 되었으니 이 자료를 대상으로 의미있는 동작(알고리즘)을 해 보겠습니다. "정수1000개.txt" 파일이 있으니 **정수의 값을 오름차순으로 정렬한 후 정렬된 결과를 화면에 출력**합니다.

정렬하는 방법은 매우 많습니다. 수천가지 이상의 정렬알고리즘이 있습니다. https://ko.wikipedia.org/wiki/정렬_알고리즘을 읽어보기 바랍니다. Youtube에서 정렬알고리즘을 키워드 sort로 검색하면 정렬과정을 동영상으로 볼 수 있습니다. 그 중 하나의 주소를 적어 봅니다. https://www.youtube.com/watch?v=y9Ecb43qw98. 가만히 보고만 있어도 재미있으며 시간이 잘 갑니다.

많은 알고리즘이 있지만 대개 C에서는 qsort, C++에서는 sort 함수로 정렬합니다. 이 함수들이 어떤 알고리즘으로 구현되어 있는지는 구글에서 찾아볼 수 있지만 사용자의 입장에서 보면 몰라도 아무런 상관이 없습니다. 다만 C++의 sort를 더 잘 이해하려면 C의 qsort를 알아보는 것이 큰 도움이 됩니다.

qsort를 먼저 설명해 보겠습니다. 알고리즘을 실행하려면 대상 자료가 먼저 준비되어 있어야 합니다. 파일에 있는 1000개의 정수를 배열로 읽어 오겠습니다.

```
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;

int main() {
    ifstream in( "정수1000개.txt", ios::binary );

    int data[1'000];

    in.read( (char*)&data, sizeof( int ) * 1'000 );

    // 확인용 출력
    for (int n : data)
            cout << setw( 20 ) << left << n;
}
```

어떻습니까? 매우 간단하죠? 이제 data에는 1000개의 정수가 저장되어 있습니다. C언어의 qsort 함수를 사용하여 정렬하겠습니다.

[실습] qsort 함수를 검색해보자. 함수의 사용법을 이해했다면 data를 정렬해 보자. (10분)

```
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;

int main() {
    ifstream in( "정수1000개.txt", ios::binary );

    int data[1'000];

    in.read( (char*)&data, sizeof( int ) * 1'000 );

    qsort( data, 1'000, sizeof( int ), compareFunc );

    // 확인용 출력
    for (int n : data)
            cout << setw( 20 ) << left << n;
}
```

정말 중요한 함수입니다. qsort는 정렬하려고 하는 것이 무엇인지 상관없이 정렬할 수 있습니다. qsort를 알고리즘 함수로 생각할 수 있는데 이 함수는 자료형을 따지지 않습니다. 기능면에서 이와같은 함수를 **제네릭(generic) 함수**라 합니다.

어떤 자료를 어떻게 정렬할 것인가는 정렬하려는 사람 마음입니다. 다시 말하면 같은 자료를 정렬하는 방법은 여러 가지가 있을 수 있다는 것입니다. 정수만 봐도 오름차순으로 정렬할 수 있고 또 내림차순으로도 정렬할 수 있는 것입니다. 지금 코로나 바이러스 때문에 이렇게 강의 자료를 만들고 있습니다만 처리해야 할 자료가 코로나 관련이라고 해 봅시다. 확진자가 많은 순서로 또는 사망자 순으로 또는 새로 늘어난 확진자 순으로와 같이 정렬기준에 따라 자료를 정렬할 수 있는 것입니다.

이러한 상황을 정렬해야 하는 함수 입장에서 생각해 봅시다. 자료를 어떤 기준으로 정렬하고 싶은 지 함수에게 알려줘야 정렬할 수 있겠죠? 자. 아주 중요한 순간입니다. **함수에게 어떻게 하라는 것을 어떻게 알려줄 수 있을까요?** 여러분이 지금까지 함수에게 전달해 준 것은 무엇입니까? 전부 어떤 값이었을 것입니다. 지금 qsort 함수에 필요한 것은 값이 아니라 정렬할 수 있는 기능입니다. 프로그램에서 기능이 코딩되어 있는 부분이 바로 함수입니다. 즉 qsort의 네 번째 인자에 전달될 것은 바로 기능이 적혀있는 함수의 시작 번지가 되어야 하는 것입니다. 물론 함수는 미리 선언되고 정의되어 있어야 되겠습니다. 제대로 코딩해 보겠습니다.

```
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;

int compareFunc( const void*, const void* );

int main() {
    ifstream in( "정수1000개.txt", ios::binary );

    int data[1'000];

    in.read( (char*)&data, sizeof( int ) * 1'000 );

    qsort( data, 1'000, sizeof( int ), compareFunc );


    // 확인용 출력
    for (int n : data)
            cout << setw( 20 ) << left << n;
}

int compareFunc( const void* a, const void* b )
{
    int num1 = *static_cast<const int*>(a);       // 문법을 잘 살펴보세요
    int num2 = *static_cast<const int*>(b);

    return num1 - num2;
}
```

어떻습니까? 오름차순으로 정렬되었나요? 정렬함수 compareFunc의 인자와 리턴형식은 그렇

게 하기로 정한 것입니다. 내림차순으로 정렬하려면 함수를 이렇게 수정하면 됩니다.

```
int compareFunc( const void* a, const void* b )
{
    int num1 = *static_cast<const int*>(a);
    int num2 = *static_cast<const int*>(b);

    return num2 - num1;
}
```

그런데 C++에서는 **이름 없는 함수** 즉, **람다(lamba)**가 추가되었습니다. 기억나시죠? 위의 정렬 코드를 살펴보면 compareFunc라는 이름의 함수는 기능만이 필요할 뿐인데 이름을 붙였습니다. 아마도 프로그래머가 다른 곳에서 이 함수를 부를(function call) 일은 전혀 없을 텐데 말이죠. 람다를 사용하여 바꿔 보겠습니다.

```
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;

int main() {
    ifstream in( "정수1000개.txt", ios::binary );

    int data[1'000];

    in.read( (char*)&data, sizeof( int ) * 1'000 );

    qsort( data, 1'000, sizeof( int ), []( const void* a, const void* b ) {
            return *(static_cast<int const*>(a)) - *(static_cast<int const*>(b));
            } );

    // 확인용 출력
    for (int n : data)
            cout << setw( 20 ) << left << n;
}
```

함수의 기능만이 필요한 곳에 사용하는 것이 람다입니다. 코드가 줄어들었습니다. 또한 앞의 코드에 비해 qsort 함수를 관찰하면 어떻게 정렬하려는 것인지 바로 알 수 있습니다. 이 코드는 이전 코드에 비해 **가독성(readability)**이 좋다고 합니다. 분명 더 좋은 코드입니다. STL에서는 알고리즘 함수의 기능을 변경하기 위해 람다를 사용합니다. 바꿔 말해 봅니다. **STL은 람다를 이용하여 알고리즘의 기능을 원하는 대로 바꿀 수 있습니다.** 람다의 사용법을 잘 알아야 합니다. 괜찮은 블로그가 많으니 찾아보기 바랍니다.

qsort 함수는 어떤 자료라도 정렬할 수 있습니다. **메모리가 연속**되어 있다면 말이죠. 인자를 살펴보면 정렬할 메모리의 시작번지와 자료 한 개의 크기 그리고 모두 몇 개인가를 알려줘야 하는 것이죠. 그러니 당연히 메모리가 물리적으로 연속된(contiguous) 자료만 정렬할 수 있는 것입니다. 우리는 정수 1000개를 배열로 읽어 왔습니다. 배열이 바로 연속된 메모리입니다. 메모리의 모습을 그려보세요. 자료가 정렬되는 모습을 상상해 보세요. **C++ 언어에서 메모리의 모습을 이해하지 못하면 제대로 프로그램할 수 없으며 프로그램이 다행히 실행된다고 하더라도 언제 어떻게 멈춰 버릴지 알 수 없습니다**.

이제 C++ 언어에서 같은 프로그램을 어떻게 하는 지 알아 보겠습니다.

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <algorithm>
using namespace std;

int main() {
    ifstream in( "정수1000개.txt", ios::binary );

    int data[1'000];

    in.read( (char*)&data, sizeof( int ) * 1'000 );

    sort( begin(data), end(data), [](  int a, int b ) {
            return a < b;
    } );

    // 확인용 출력
    for (int n : data)
            cout << setw( 20 ) << left << n;
}
```

훨씬 간결합니다. 정렬할 자료의 범위를 알려주고 람다로 어떻게 정렬할 지 지시합니다. 내림차순으로 정렬하려면 부등호의 방향만 바꾸면 됩니다.

```
    sort( begin(data), end(data), [](  int a, int b ) {
            return a > b;
    } );
```

POD 타입을 어떻게 다루는지 익숙해졌을 것입니다. 과제를 제시하며 이번 강의를 마치겠습니다.

[2주 1일 - 과제]

정수 1000개는 시시하다. 1000만개 정도는 다루어야 컴퓨터가 일 좀 할 것 같다. 누군가가 정수 1000만개를 다음과 같이 파일 "정수1000만개.data"에 저장하였다. 우리는 이 소스파일은 모르는 상태이다. 단지 파일 "정수1000만개.data"를 전달받았다고 가정하자. 파일 크기가 정확히 40,000,000 바이트인가 확인하자.

```
#include <iostream>
#include <random>
#include <fstream>
using namespace std;

int main() {
    default_random_engine dre;
    uniform_int_distribution uid;

    ofstream out( "정수1000만개.data", ios::binary );

    int n;

    for (int i = 0; i < 1000'0000; ++i) {
            n = uid( dre );
            out.write( (char*)&n, sizeof( n ) );
    }
}
```

[문제 1] 파일에 저장한 정수 1000만개를 읽어 와라.
[문제 2] 읽은 1000만개의 정수를 C++의 sort를 사용하여 정렬하라.
[문제 3] 정렬한 정수를 "정수1000만개정렬.data"로 저장하라.

각 문제를 하나의 소스파일에 코딩하여 해결할 수 있을 것이다.
이 문제를 해결하지 못한 학생은 어떻게 하려고 했고 어디에서 잘못되었는지 파악한 결과를 제출하면 된다.

2주 1일과 2주 2일 과제를
문서 파일 하나로 만들어 (형식은 윈도우 텍스트 문서 형식으로)
3주 1일 강의 시작 전까지 e-class로 제출해 주세요.

과제는 **친구들과 의논하여 해결하는 것을 권장**합니다.
과제를 평가하여 앞으로 얼마나 공부해야 할 것인지 알려주겠습니다.