STL - 3주 1일

Wulong

3주 강의 2020. 3. 30 ~ 2020. 4. 3	
지난 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
이번 주 3.30 ~ 4.3	**객체지향 프로그래밍과 제네릭 프로그래밍** x86 프로세서의 메모리 구조 다수의 클래스 객체를 사용하는 프로그램 작성 호출가능 타입 스마트 포인터
다음 주 4.6 ~ 4.10	**순차 컨테이너** 자원을 확보하는 컨테이너 만들어 두기 array vector

지난 주

안녕하세요? STL 수강생 여러 분!
한 주가 또 지났습니다. 2주 강의 재미있게 공부하셨나요? 내용이 조금 어렵지 않을까 걱정도 하였습니다마는 질문이 없어 적절한 수준이었나보다라고 생각하였습니다. 빨리 강의실에서 만나고 싶지만 현재 상황으로 봐선 언제 강의실에서 만날 수 있을지 모르겠습니다.

직접 강의하는 것이 훨씬 더 재미있고 쉬운데, 없는 자료를 만들려니 시간도 많이 걸리고 힘도 들면서 신나지도 않으니 답답하기만 합니다. 그래도 직접 강의하는 것처럼 진짜로 알아야 할 것들을 편한 말투로 적어보려고 애쓰고 있습니다. 쉽게 읽히도록 말이죠. 물론 전문적인 내용이기 때문에 쉽게 읽긴 어려울 것입니다. 어쨌든 나름 노력하겠습니다.

과제 제출기한을 다음 주 강의 시작 전으로 해 놓고 기한이 지난 후 읽어보기 시작하였습니다. **과제의 평가 점수는 0점**으로 해 놓았습니다. 학점에 반영하지 않겠다는 뜻 입니다. 그러니 편하게 서로 의논하면서 본인이 이해한 것을 정리하여 제출하면 되겠습니다. 90% 이상의 학생들이 텍스트 파일로 제출하여 쉽게 읽어볼 수 있었습니다. 과제를 보면서 거의 모든 학생들이 STL을 수강하는 데 큰 문제없을 것이라 판단하였습니다. 과제는 클래스의 멤버와 입출력 함수를 만들 수 있으면 그냥 해결할 수 있는 문제여서 잠깐 시간을 들여 코딩해보면 실행되게 할 수 있었을 것입니다. 그런데 동영상을 만들어 보기로 했던 것은 반 이상의 학생들이 멤버 변수를 public으로 선언하여 문제를 풀었기 때문이었습니다. 멤버변수를 public으로 하면 class라고 쓰지 말고 struct라고 써야 합니다. 그렇게 한 학생은 없었습니다. 동영상으로 설명한 것과 같이 private으로 선언하면 생성자를 만들지 않고는 문제를 해결할 수 없습니다. 앞으로도 글로 부족한 부분이 있다면 가끔 동영상을 만들어 올리겠습니다.

책을 읽어보기 바랍니다. 템플릿을 설명하지 않는 C++ 책은 없을 것입니다. 템플릿 부분을 읽고 무슨 내용인지 모르더라도 상관없습니다. 공부가 어려운 것은 익숙하지 않기 때문입니다. 일단 한 번 읽는 것이 중요합니다. 다음에 또 읽어 보세요. 그리고 다음에 또 읽어 보는 겁니다. 옛날 사람들은 “독서백편의자현(讀書百遍意自見)”이라는 말을 따라 깨우칠 때까지 계속 책을 봤습니다. 지금은 그렇게까지 할 필요는 없습니다. 옛날엔 책이 정말 귀했습니다. 그러니 같은 책을 계속 볼 수밖에 없었죠. 지금은 이 책 저 책 구글 유튜브 등 같은 주제를 다룬 여러 정보를 살펴보면 됩니다. 같은 템플릿이라도 다른 설명을 보면 자기에게 더 잘 이해되는 것이 있을 수 있을 거거든요. 물론 그렇게 반복하기 때문에 시간이 지날수록 확실하게 알 수 있게 되는 것이겠죠. 여러 번 본다는 것이 정말 중요합니다. 뇌는 반복되는 입력정보를 더 오래 기억할 수 있는 장기 기억장소로 옮겨 저장한다고 합니다.

우리 **책 1장에 이번 학기 배울 내용 전체가 요약 소개되어 있습니다**. 읽어서 이해되는 학생은 이 강의를 들을 필요 없습니다. 1장이 쉽지 않다는 말입니다. 그래도 읽어 보세요. 22쪽의 학습할 내용을 살펴보세요. 참고로 지난 시간 알고리즘 시간을 재는 내용은 이 책 632쪽에 있습니다. 내가 정리한 내용과 비교해 보세요. 아마 내 강의를 더 열심히 공부하고 싶어질 겁니다.

2주 과제 1과 2는 과제 마감 이후에 해설 동영상을 올리겠습니다.

객체지향 프로그래밍과 제네릭 프로그래밍

객체지향이라는 개념을 어떻게 이해하고 있나요? 객체지향이라는 것은 말 그대로 객체 위주로 코딩해서 문제를 해결한다는 뜻입니다. 현실 세계의 문제를 프로그램으로 모사하거나 해결하기 위해 키워드 class로 대상 객체를 모델링합니다. 수치로 표현할 수 있는 특성만이 아니라 모델 대상의 행동을 나타내는 함수를 하나의 자료형으로 함께 묶는 것입니다. 그럴 필요가 있다면 새로 만든 자료형을 기존 자료형과 유사한 방식으로 다룰 수 있도록 연산자를 오버로딩하기도 합니다. 다른 자료형과 다른 **나만의 독특한 자료형을 만들어 문제를 해결**한다는 것이 중요합니다. 클래스의 멤버(함수와 변수)는 디폴트로 다른 자료형을 갖는 객체는 사용할 수 없습니다(class의 모든 멤버는 default private임). 특히 멤버 변수를 그렇게 만듭니다. 멤버 함수는 바깥세상과 의사전달을 위한 창구의 역할을 합니다. 대부분의 멤버함수는 인터페이스 함수로 기능합니다. 클래스의 생성과 소멸에 관계된 스페셜 함수들은 특별한 경우를 제외하고는 public으로 지정합니다. 객체지향 프로그래밍이라는 개념에서 중요한 것은 내가 만든 객체는 어떤 **역할**을 어디까지 **책임**지고 수행할 것인가? 다른 객체와는 어떤 **관계**를 갖고 어떻게 의사소통을 하도록 만들 것인가를 설계하는 것입니다.

하나의 자료형에 집중하는 객체지향 프로그래밍과 달리 **제네릭 프로그래밍**은 한마디로 **자료형에 무관한 프로그래밍 개념**이라고 말할 수 있습니다. 1주와 2주 강의의 change 함수를 생각해 보면 되겠습니다. 1주 강의의 change 함수는 두 개 int 자료형의 메모리 내용을 서로 바꾸거나 클래스 X 자료형의 메모리 내용을 서로 바꾸는 함수였습니다. 2주 강의의 change 함수는 두 인자의 자료형이 서로 같다면 자료형에 상관없이 두 자료형의 메모리 값을 바꿀 수 있었습니다. 어떻게 그렇게 할 수 있었죠? 네. 키워드 template을 사용하여 template 함수를 만들고 실제 특정 자료형을 인자로 갖는 함수는 컴파일러가 만들도록 하면 되는 것이었습니다. C++ 언어에서

- **객체지향 프로그래밍의 핵심 키워드는** class
- **제네릭 프로그래밍의 핵심 키워드는** template

아주 간단하게는 위와 같이 정리할 수 있겠습니다.

C++ 언어 안에는 서로 다른 프로그래밍 개념이 있습니다. 특히 객체지향과 제네릭이라는 개념은 언뜻 보기에 전혀 다른 프로그래밍 개념으로서 이 두 개념이 서로 어울릴 일은 없어 보입니다. 하나는 특별한 자료형을 만들어 문제를 해결하려는 것이고 다른 하나는 자료형에 무관하게 문제를 해결할 수 있다고 주장하는 것이니까요. 둘이 어울릴 수 있는 곳은 어디일까요? 답은 **자료구조**와 **알고리즘**에 있습니다. 먼저 자료구조와 알고리즘이 처리할 것들이 무엇인가 생각해 보세요.

자료구조는 많은 수의 자료를 다룹니다. 자료의 수가 적다면 어떤 자료구조를 사용하더라도 자료를 처리하는데 필요한 자원(시간과 공간)에 큰 차이가 나지 않습니다. 자료구조는 특정한 자료형 만을 대상으로 하지 않습니다. 처리할 자료형이 동일하기만 하다면 아무 문제 없습니다. 알고리즘도 같습니다. **알고리즘은 자료구조에서 있는 자료 중에서 원하는 자료를 원하는**

방식으로 처리합니다. 어떤 함수 f가 int 자료형을 원소로 갖는 자료구조 스택에서 가장 큰 값을 제거한다고 해보죠. 함수 f를 알고리즘이라고 할 수 있을까요? 무엇을 알고리즘이라고 하는가는 이전 학기에 공부했던 알고리즘 강의에서나 전문가들이 연구해 놓은 정의를 살펴보면 좋겠습니다. 알고리즘은 문제를 해결하기 위한 절차를 기술한 것이며 특정 자료형을 대상으로 하지는 않는다는 것을 알 수 있을 것입니다.

STL은 동일한(homogeneous라는 단어를 씁니다) 자료형의 객체가 많이 있는 **자료구조**에서 목적에 맞는 자료를 처리할 수 있는 **알고리즘** 함수를 제공하는 라이브러리입니다. 쉽게 생각해 보면 내가 class로 만든 자료가 많이 있는 데 이것을 처리할 수 있는 자료구조와 알고리즘이 제공된다는 것입니다. 내가 만들 게임에 참여한 Player 1‘000명이 있다고 가정해 보죠. 각 Player를 아이디 순서대로 정렬하거나, 점수 순서대로 위에서 10명을 추리거나 할 일이 있을 수 있겠죠? 아니면 수 천 개의 탄알로 탄막을 형성했는데 내 비행선이 탄알에 맞았는지 아닌지 판단해야 할 때도 있을 것입니다. 이와 같은 일이 필요할 때 **자료구조**와 **알고리즘**을 프로그래머가 매번 만들지 않아도 되도록 하는 것이 STL의 주 역할 되겠습니다.

X86 프로세서의 메모리 구조

그럼 이제 지난 주 과제를 되돌아보는 것이 의미 있겠습니다. 2주 1일 과제는 정수 1‘000만 개를 정렬해 보라는 것이었습니다. 1’000개가 아니고 1‘000만개입니다. 과제를 하면서 정수 1’000만개를 지역변수로 만들 수 없다는 것을 알았을 것입니다. 이 과제는 **변수**(변수란 메모리를 차지하고 있는 읽고 쓸 수 있는 객체를 말합니다)**가 있어야 할 곳은 어디인가**라는 것을 묻고 있는 것입니다.

이번 내용의 제목을 거창하게 붙여 봤습니다만 하고 싶은 이야기는 변수가 있을 수 있는 메모리 장소는 다음 3곳 중 하나여야 한다는 것입니다.
- STACK
- DATA
- HEAP

프로그램이 실행될 때 메인 메모리의 일부분을 차지하게 되는 데 이 메모리는 크게 CODE, DATA, STACK, HEAP이라는 영역으로 구분됩니다.

[실습] 지금 바로 CODE, DATA, STACK, HEAP을 검색하기 바랍니다. (10분)

정수 1‘000만개를 저장하기 위한 공간은 최소 1’000만 × sizeof(int) 바이트입니다. 여기에서 최소라는 표현을 한 것은 다른 자료구조를 사용하면 더 큰 메모리 공간을 사용하여야하기 때문입니다. 최소한의 크기를 사용하면서 정수 1‘000만개를 저장하는 방법은 물리적으로 연속된 메모리 공간을 사용하는 것입니다. 그렇습니다. 바로 배열을 사용하는 것입니다. 계속 말이 너무 길었는데 실제 수업에서 제가 이렇게 하지는 않습니다. 이렇게 하면 바로 학생들 다 잡니다. 일단 코딩해 보겠습니다.

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

// 파일에서 1000'0000개의 int를 읽는다.

int main()
{
    string file{ "정수1000만개.data"s };

    ifstream in( file, ios::binary );
    if ( !in )
            cout << file << "을 열지 못했습니다" << endl;

    int n;

    for (int i = 0; i < 1000'0000; ++i) {
            in.read( (char*)&n, sizeof( int ) );
            if (i % 1'0000 == 0)
                    cout << i << "번째 int " << n << endl;
    }
}
```

파일에서 자료를 읽을 때는 파일이 지정한 폴더에 있나와 같이 파일을 여는데 문제가 없는지를 먼저 검사해야 합니다. 위 코드의 if (!in)과 같이 검사하면 됩니다. 이 프로그램은 저장된 값을 읽는 것은 문제없겠지만 읽은 값을 저장하지는 못합니다. 자료를 저장할 수 있는 곳은 3군데라고 했습니다. 각각 알아보겠습니다.

```
// 생략

// 파일에서 1000'0000개의 int를 STACK 변수로 읽는다.

int main()
{
    string file{ "정수1000만개.data"s };

    ifstream in( file, ios::binary );
    if ( !in )
            cout << file << "을 열지 못했습니다" << endl;

    int n[1000'0000];
    in.read( (char*)&n, 1000'0000 * sizeof( int ) );

    cout << "한 번의 read 함수호출로 읽는다" << endl;
}
```

어떤 변수가 지역에서 만들어졌다면 그 변수는 STACK에 생성된 것입니다. C++ 코드에서 **지역이란 { 과 } 사이**를 말합니다. 이 프로그램을 실행해 보세요. 문제없이 실행되나요? 아! 잠깐만요. 코드를 붙여넣기 해서 실행하면 안됩니다. 실력이 늘지 않습니다. 타이핑해 주세요. 실행화면에 마지막 cout 문장이 나와야 실행된 것입니다. 문장이 안 나오죠? 지역변수 n은 40MB 메모리의 시작번지를 저장하고 있는데 지역에 이 정도 크기의 메모리를 만들 수는 없습니다(방법이 없는 것은 아닙니다). 해결방법은 n을 전역변수로 만드는 것입니다.

```
// 생략

int n[1000'0000];

// 파일에서 1000'0000개의 int를 DATA 영역에 있는 변수로 읽는다.

int main()
{
    string file{ "정수1000만개.data"s };

    ifstream in( file, ios::binary );
    if ( !in )
            cout << file << "을 열지 못했습니다" << endl;

    in.read( (char*)&n, 1000'0000 * sizeof( int ) );

    cout << n[0] << endl;
    cout << n[1000'0000 - 1] << endl;
}
```

40MB의 메모리를 전역으로 확보하는데 아무런 문제가 없습니다. 프로그램을 실행하자마자 화면 출력하고 종료되는 것을 보면 이 정도 읽는 것은 순식간에 이루어지는 것 같습니다. 이제 데이터가 준비되었으니 정렬과 같은 원하는 동작을 하는 것도 문제없을 것입니다. 그런데 변수가 존재할 수 있는 곳이 한 곳 더 있었습니다. 바로 HEAP입니다.

변수가 DATA 영역에 있으면 어떤 문제가 있을까요? 문제가 될 수도 있고 안 될 수도 있지만 DATA 영역에 있는 변수는 프로그램이 실행되는 동안 언제나 그 메모리를 차지하고 있다는 것을 알아야 합니다. 프로그램이 실행된다는 것은 컴퓨터의 중요 자원인 메모리를 차지한다는 것이고 한 프로그램이 메모리를 많이 사용한다면 전체 시스템의 가용 메모리 크기가 줄게 됩니다. 이러한 점은 프로그램의 성격에 따라 항상 메모리를 차지하는 것이 꼭 필요할 수가 있기 때문에 문제가 될 수도 있고 안 될 수도 있다고 표현한 것입니다. 생각을 달리해서 프로그램이 실행되는 전체 시간동안이 아니라 일정한 시간에만 큰 메모리가 필요한 경우를 생각해 봅시다. 필요한 시점에 메모리를 확보한 후 사용이 끝난 후 돌려준다면 전체 컴퓨터의 메모리를 더 효율적으로 사용할 수 있을 것입니다. 이런 경우 메모리를 확보하고 되돌려 주는 방법이 필요한 데 C++ 언어에서는 키워드 new와 delete로 그렇게 합니다. 코드를 보시죠.

```
// 생략

// 파일에서 1000'0000개의 int를 HEAP 영역에 확보한 메모리로 읽는다.

int main()
{
    string file{ "정수1000만개.data"s };

    ifstream in( file, ios::binary );
    if ( !in )
            cout << file << "을 열지 못했습니다" << endl;

    int* p { new int[1000'0000] };

    in.read( (char*)p, 1000'0000 * sizeof( int ) );

    // 데이터를 원하는대로 사용한다.

    cout << p[0] << endl;
    cout << p[1000'0000 - 1] << endl;

    delete[] p;                         // 사용 후 반드시 반납한다.
}
```

C++ 언어가 다른 언어와 다른 부분이 여기입니다. 바로 프로그래머가 필요한 시점에 마음대로 메모리를 확보하고 사용할 수 있다는 점입니다. 최근에 개발된 프로그래밍 언어들은 메모리를 사용자가 마음대로 건드릴 수 없게 하여 안정성을 확보하였다라고 자랑합니다. 프로그래머가 메모리를 마음대로 다룰 만큼 똑똑하다고 생각하지 않고 있는 것입니다. 컴퓨터가 어떻게 구성되어 있는지 CPU와 메모리가 어떻게 작동하는지 운영체제가 모든 것을 어떤 식으로 지휘하는지 배울 만큼 배운 여러분과 같은 프로그래머에게는 기분 나빠야 할 일입니다.

C++ 언어는 프로그래머를 신으로 생각합니다. C++ 언어를 사용하는 프로그래머는 컴퓨터를 매우 잘 이해하고 있으며 메모리를 마음대로 조작하는 것쯤은 숨 쉬는 것과 같이 자연스럽게 할 것이라 생각합니다.

HEAP 영역에 메모리를 확보하는 것은 STACK이나 DATA와는 다른 점이 있습니다. 그것은 바로 변수의 도움 없이 단독으로 메모리를 확보한다는 것입니다.

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

int main()
{
```

```
    // 메모리를 계속 확보해 본다
    size_t memSize{ 100'000'000 };              // 100MB

    while (true) {
            new char[memSize];
            cout <<  memSize << "바이트 확보" << "\n";
            this_thread::sleep_for( 100ms );
    }
    cout << "이 문장이 출력될 리 없다" << endl;
}
```

한 번에 100MB씩 메모리를 확보하는 프로그램입니다. new char[memSize];라는 단독 문장으로 메모리를 확보합니다. 계속 확보합니다. 언제까지 가능할까요? 아마 수 십 번 화면 출력되고 프로그램이 비정상 종료될 것입니다. 프로그램을 수정하여 확보한 메모리를 되돌려 주면 이 프로그램은 영원히 실행될 것입니다. 이렇게 말이죠.

```
int main()
{
    // 메모리를 계속 확보해 본다
    size_t memSize{ 100'000'000 };              // 100MB

    while (true) {
            char* p = new char[memSize];
            cout <<  memSize << "바이트 확보" << "\n";
            this_thread::sleep_for( 100ms );
            delete[] p;
    }

}
```

혹시 이 코드를 몇 번 실행하다가 시스템이 바이러스가 검출되었다고 할지도 모릅니다. 걱정할 것은 없습니다. new로 확보한 메모리를 사용하려면 거기가 어딘지 기록해 놔야 합니다. 메모리를 기록한다는 것은 메모리의 번지를 적어 놓아야 한다는 것이고 이런 용도의 변수를 **포인터**라고 하는 것입니다. 위 프로그램에서 포인터 p는 STACK 영역에 자리한 지역변수임을 잊지 말아 주세요. p를 이용해 사용한 후 반납할 때도 p가 필요하다는 것도 잊지 마세요.

마지막으로 집고 넘어갈 것이 있습니다. **크기가 큰 자료를 DATA에 둘 것인지 HEAP에 만들 것인지에 대한 정답은 없습니다.** HEAP에 생성한 자료는 공간을 효율적으로 이용한다고 생각하겠지만 그에 따른 단점이 있습니다. 바로 시간입니다. new와 delete에 소요되는 시간은 보통 명령들과는 성격이 매우 다릅니다. 이와 같은 코드를 시간이 생명인 게임 루프내에서 실행한다면 그 게임은 영원히 출시하지 못합니다. 반대로 DATA 영역에 자리한 변수는 시간에서 손해볼 것이 없다는 점이 HEAP 영역에 비해 장점이라고 할 수 있겠습니다. 프로그램의 성격에 맞게 정해 사용하면 되는 것입니다. 이 둘의 장점을 결합한 사용자 정의 new/delete 기법도 있다는 것을 적어두며 마치겠습니다.

다수의 클래스 객체를 사용하는 프로그램

이번 강의는 설명 위주로 진행하느라 나는 정말 고생했고 학생들은 지루했겠지만 필요한 내용은 다 설명했습니다. 강의하면서 강조하는 것 중 하나는 "예제와 같이 int를 사용하는 실제 프로그램은 없다"입니다다. 책에 예로 나오는 char, int와 같은 POD 타입으로 프로그램 가능하다면 객체지향 개념 같은 것은 배울 필요도 없을 것입니다.

이제 다수의 객체를 사용한 프로그램으로 지금까지 강의 내용을 반복학습해 보겠습니다. 공부는 반복 반복 또 반복입니다.

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

int gid { };

class Dog  {
    string name;                    // 이름 - 15글자까지만 허용
    int age;                        // 나이
    int id;                         // 생성 시 결정되는 고유의 숫자로 된 id

public:

};

int main()
{
    // cout << "Dog의 메모리 크기 - " << sizeof( Dog ) << endl;

    Dog dog{ "댕댕이", 3 };          // id는 객체생성 시 ++gid 값으로 결정

    cout << dog << endl;            // 댕댕이, 3, 1
}
```

여기에서 시작합니다. 모두 시간을 들여 실습하며 진행해 주세요. 실습 답은 설명하면서 바로 제시할 것이니 부담 갖지 말고 코딩하세요.

[실습] 주석을 풀어 객체의 크기를 확인하라. (5분)

자료형의 이름이 Dog입니다. sizeof(Dog)로 이 자료형이 차지하게 될 메모리의 크기를 알 수 있습니다. 컴파일러는 메모리의 크기와 메모리의 접근 속도 사이에서 어떻게 Dog의 메모리를 구성할 지 결정할 수 있습니다. 사용자가 어떻게 하라고 알려주기만 한다면 말이죠. 사용자는 원한다면 Dog의 메모리 크기를 얼마(2의 배수가 되어야 함)로 하라고 지시할 수 있습니다. 예를 들어 볼까요?

```
class alignas(64) Dog  {
    string name;                    // 이름 - 15글자까지만 허용
    int age;                        // 나이
    int id;                         // 생성 시 결정되는 고유의 숫자로 된 id

public:

};
```

Dog의 크기를 다시 확인해 보세요. 이런 건 몰라도 됩니다. 중요한 것이 아니에요.

[실습] main이 수정 없이 실행되도록 class를 코딩하라. (10분)

10분이면 적당할 겁니다. 내 코드 말고 **직접 생각해보는 것이 중요**합니다. 생각할 수 있는 모든 것을 찾아보세요. 머릿속에 정보가 있다면 가장 빠를 것입니다. 반복해서 공부했다면 장기기억저장소에 이 내용이 잘 저장되어 있을 겁니다. 그렇지 않더라도 어디에서 정보를 찾을 수 있을 것이다라는 것을 안다면 무엇을 어떻게 해야 할지 모르는 것보다 빨리 이 문제를 해결할 수 있을 겁니다. 내 답은 이렇습니다.

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

int gid { };

class Dog  {
    string name;                    // 이름 - 15글자까지만 허용
    int age;                        // 나이
    int id;                         // 생성 시 결정되는 고유의 숫자로 된 id

public:
    Dog( string name, int age ) : name{ name }, age{ age }, id( ++gid ) {
    }

    friend ostream& operator<<( ostream&, const Dog& );
};

int main()
{
    // 생략
}

ostream& operator<<( ostream& os, const Dog& dog )
{
    os << dog.name << ", " << dog.age << ", " << dog.id;
```

```
    return os;
}
```

제목이 **다수의 객체를 사용하는 프로그램**임을 잊지 않았죠? 이제 객체 한 개를 만들어 본 겁니다. 사실 한 개를 만들면 여러 개 만드는 것은 쉽잖아요? 지금까지와 같이 배열을 사용한다면 그냥 Dog [10000]; 이라고 쓰면 10000마리의 Dog를 만들 수 있으니 말이죠. 그런데 이렇게 객체를 만들려면 디폴트 생성자가 필요하단 말이죠? 디폴트 생성자에서 이름, 나이, 아이디를 만들도록 하겠습니다. 이렇게요.

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

int gid { };

default_random_engine dre;
uniform_int_distribution<int> uidAge( 1, 12 );
uniform_int_distribution<int> uidName( 'a', 'z' );
uniform_int_distribution<int> uidNameLen( 3, 15 );

class Dog  {
    string name;            // 이름 - 15글자까지만 허용
    int age;                // 나이
    int id;                         // 생성 시 결정되는 고유의 숫자로 된 id

public:
    Dog(): id{ ++gid } {
            int len = uidNameLen( dre );
            for (int i = 0; i < len; ++i)
                    name += uidName(dre);
            age = uidAge( dre );
    }

    Dog( string name, int age ) : name{ name }, age{ age }, id( ++gid ) {
    }

    friend ostream& operator<<( ostream&, const Dog& );
};

int main()
{
    Dog dogs[100];

    for ( const Dog& dog : dogs)
            cout << dog << endl;
}
```

이제 많은 수의 Dog 객체를 만들 수 있게 되었습니다. 파일에 저장하는 것도 아무런 문제가 없습니다.

```
// Dog 1'0000 객체의 정보를 파일에 기록한다

int main()
{
    ofstream out( "Dog만마리", ios::binary );

    for (int i = 0; i < 1'0000; ++i) {
            Dog d;
            out.write( (char*)&d, sizeof( Dog ) );
    }
}
```

파일이 제대로 만들어졌는지 크기를 확인해 보세요. 파일 내용을 그냥 읽을 수는 없습니다. 이제 과제를 제시하고 이번 강의를 마치겠습니다. 실습과제는 공부해보라고 내는 겁니다. 점수에 들어가지 않습니다. 나는 강조합니다. **공부는** 여러분 **마음속에** 있는 겁니다. 누가 하라고 해서 하는 것이 아닙니다. 하고 싶을 때 해야 되고 그때 성과가 나는 것이 공부입니다.

그런데 말이죠! *C++은 정말 재미있습니다*. 내가 만든 이 강의록만 봐도 그렇지 않습니까?
이거 정말 재미있는 공부입니다. 알수록 재미있습니다. 지금이 바로 할 때 입니다.
그러려고 이 강의 신청하셨잖아요. 여러분!

[3주 1일 - 과제]

[문제 1] 파일에 저장한 Dog 객체 1‘0000개를 읽어 와라.
[문제 2] 읽은 Dog 객체들을 C++의 sort를 사용하여 이름 오름차순으로 정렬하라.
- 필요하다면 Dog에 getter를 코딩할 수 있다.

[문제 3] 정렬한 객체 중 제일 처음 객체와 마지막 객체를 화면 출력하라.
- 화면에 출력된 객체 정보를 따로 적어 제출하라.

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

3주 1일과 3주 2일 과제를
문서 파일 하나로 만들어 (형식은 윈도우 텍스트 문서 형식으로)
4주 1일 강의 시작 전까지 e-class로 제출해 주세요.
과제는 **친구들과 의논하여 해결하는 것을 권장**합니다.