인트아이 C++ 심화 스터디

FUNCTOR, LAMBDA, CLOSURE

2회차 퀴즈 답안

```cpp
unsigned long long fibonacci(int n) {
    if (n < 2)
        return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}

void bench() {
    BenchGuard _;
    fibonacci(25);
}
```

```
Elapsed time: 0.0005593 seconds
```

배열을 내림차순으로 정렬해줘

```cpp
#include <algorithm>
#include <iostream>

bool compare(int a, int b) {
    return a > b;
}

int main() {
    int A[10] = { 3, 6, 5, 8, 2, 1, 0, 7, 9, 4 };

    // 오름차순
    std::sort(A, A + 10);
    for (int i : A) {
        std::cout << i << ' ';
    }
    std::cout << '\n';

    // 내림차순: Function Pointer
    std::sort(A, A + 10, compare);
    for (int i : A) {
        std::cout << i << ' ';
    }
    std::cout << '\n';
    return 0;
}
```

```
0 1 2 3 4 5 6 7 8 9
9 8 7 6 5 4 3 2 1 0
```

```cpp
template< class RandomIt, class Compare >
constexpr void sort( RandomIt first, RandomIt last, Compare comp );
```

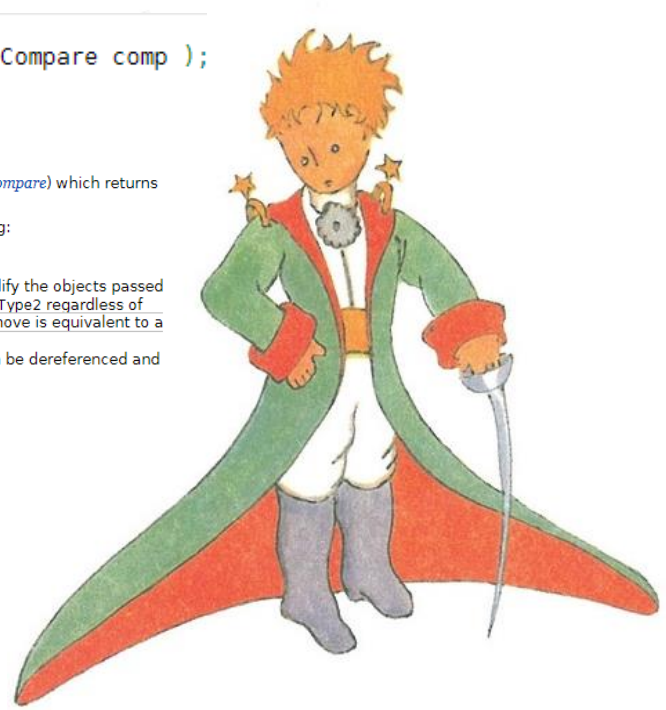
Parameters

first, last	-	the range of elements to sort
policy	-	the execution policy to use. See execution policy for details.
comp	-	comparison function object (i.e. an object that satisfies the requirements of *Compare*) which returns `true` if the first argument is *less* than (i.e. is ordered *before*) the second.

The signature of the comparison function should be equivalent to the following:

```cpp
bool cmp(const Type1 &a, const Type2 &b);
```

While the signature does not need to have `const&`, the function must not modify the objects passed to it and must be able to accept all values of type (possibly const) `Type1` and `Type2` regardless of value category (thus, `Type1&` is not allowed, nor is `Type1` unless for `Type1` a move is equivalent to a copy (since C++11)).
The types `Type1` and `Type2` must be such that an object of type `RandomIt` can be dereferenced and then implicitly converted to both of them.



https://en.cppreference.com/w/cpp/algorithm/sort

더 빠르게 할 수는 없을까

std::sort의 마지막 인자는 정렬 순서를 결정하는 비교 함수

하지만, 함수 포인터로 들어온 compare 함수는 일반적으로 inline 되지 않음

함수의 inline 여부는 성능에 중대한 영향을 미침

컴파일러가 함수를 inline할 수 있게 유도해, 성능을 끌어 올려보자

```
int addmul(int a, int b, int c) {
    return a + (b * c);
}

int main() {
    int i = addmul(1, 2, 3);
    return 0;
}
```



```
int main() {
    int i = 1 + (2 * 3);
    return 0;
}
```

inline

컴파일러가 자주 사용되는 간결한 함수 코드를, 해당 함수 호출 코드에 복사-붙여넣기 하여 함수 호출 생략을 통해 성능을 끌어올리는 기법

Functor [펑터]

=함수 객체(=함자) : 함수인 척하는 객체

객체이기 때문에 struct나 class 문법을 이용

함수처럼 동작해야 하므로 () 연산자를 오버로딩

함수 객체 클래스 자체를 타입으로 활용 가능 → C++ 템플릿과 결합하여 사용 가능

포인터 형태가 아니기 때문에 Function Pointer보다 inline 될 가능성이 높음

(당연하게도 항상 inline 되는 것은 아님, 함수의 크기가 작고 inline을 통해 성능이 오를 수 있다고 판단될 때만 inline)

Functor vs Function

Functor	Function
struct 또는 class로 생성된 객체	함수
멤버 변수를 이용해 각 Functor마다 다른 값을 저장할 수 있음	전역 변수를 이용해 값을 저장
inline 가능(더 빠르게 동작)	함수 포인터는 inline 불가

배열을 내림차순으로 정렬해줘

```cpp
#include <algorithm>
#include <iostream>

struct Compare {
    bool operator()(int a, int b) const {
        return a > b;
    }
};

int main() {
    int B[10] = { 1, 0, 2, 5, 8, 9, 4, 7, 3, 6 };

    // 내림차순: Functor
    std::sort(B, B + 10, Compare());
    for (int i : B) {
        std::cout << i << ' ';
    }
    std::cout << '\n';
    return 0;
}
```

```
9 8 7 6 5 4 3 2 1 0
```

STL에서의 Functor

functional 헤더에 std::greater와 std::less 등 다양한 비교 Functor가 존재

```
#include <algorithm>
#include <functional>
#include <iostream>

int main() {
    int B[10] = { 1, 0, 2, 5, 8, 9, 4, 7, 3, 6 };

    // 내림차순: STL Functor
    std::sort(B, B + 10, std::greater<int>());
    for (int i : B) {
        std::cout << i << ' ';
    }
    std::cout << '\n';
    return 0;
}
```

```
9 8 7 6 5 4 3 2 1 0
```

https://en.cppreference.com/w/cpp/header/functional

Member functions

`operator()`	checks whether the first argument is *greater* than the second (public member function)

std::greater::operator()

```
bool operator()( const T& lhs, const T& rhs ) const;                (until C++14)
constexpr bool operator()( const T& lhs, const T& rhs ) const;      (since C++14)
```

Checks whether lhs is *greater* than rhs.

Parameters

`lhs, rhs` - values to compare

Return value

For T which is not a pointer type, `true` if `lhs > rhs`, `false` otherwise.

For T which is a pointer type, `true` if lhs succeeds rhs in the implementation-defined strict total order, `false` otherwise.

Exceptions

May throw implementation-defined exceptions.

Possible implementation

연산자 오버로딩을 이용해 구현

```
constexpr bool operator()(const T &lhs, const T &rhs) const
{
    return lhs > rhs; // assumes that the implementation uses a flat address space
}
```

Lambda [람다]

=람다 함수(=익명 함수)

C++11에서 새로 추가된 문법 (functor 대체)

struct나 class를 선언할 필요가 없음

함수 포인터와 다르게, inline이 가능

```cpp
#include <algorithm>
#include <iostream>

int main() {
    int C[10] = { 5, 8, 2, 4, 7, 0, 6, 3, 9, 1 };

    // 내림차순: Lambda
    std::sort(C, C + 10, [](int a, int b) {
        return a > b;
    });
    for (int i : C) {
        std::cout << i << ' ';
    }
    std::cout << '\n';

    return 0;
}
```

Lambda

대괄호, 소괄호, 중괄호를 이용해 람다 함수를 정의 → 코드가 뜬금없이 [] 으로 시작한다면 람다 함수

```
auto addmul = [](int a, int b, int c) {
    return a + b * c;
};
int i = addmul(1, 2, 3);
```

중괄호 안에는 함수의 동작을 정의

소괄호 안에는 함수의 인자를 정의

람다 함수의 이름은 변수를 통해 정의함.

람다 문법 자체적으로 함수의 이름을 지정하는 기능이 없음 → 익명 함수라고 불리는 이유

```
auto addmul = [](int a, int b, int c) -> int {
    return a + b * c;
};
```

람다 함수의 반환 타입을 명시할 수도 있음

Lambda

```
int main() {
    int n = 5;
    auto addn = [](int a, int b) {
        return a + b + n; // error
    };
    int j = addn(1, 2);
    return 0;
}
```

main 함수 안에 정의된 n을 addn 람다 함수에서 사용하려 했지만 컴파일 오류 발생

main과 addn은 개별적인 함수 스코프를 갖기 때문에 addn에서는 n에 접근할 수 없음

Closure

람다 함수 내부에서 외부의 값에 접근할 수 있게 하는 방법

외부의 변수를 Capture하여 람다 함수에 풀어 줌

```
int main() {
    int n = 5;
    int m = 5;
    auto addn = [n](int a, int b) {
        return a + b + n;
    };
    int j = addn(1, 2);
    return 0;
}
```

외부의 n을 Copy 방식으로 Capture

[n]	n을 복사하여 Capture
[n, m]	n과 m을 복사하여 Capture
[&n]	n을 참조 방식으로 Capture
[=]	외부의 모든 변수를 복사하여 Capture
[&]	외부의 모든 변수를 참조 방식으로 Capture

Quiz

주어진 코드를 참고하여
홀수를 먼저 오름차순으로
출력하고,
짝수를 내림차순으로
출력하는
우선순위 큐를 구현하시오.
(Functor를 이용할 것)

https://en.cppreference.com/w/cpp/container/priority_queue

```cpp
#include <functional>
#include <iostream>
#include <queue>
#include <vector>

int main() {
    int A[10] = { 3, 6, 5, 8, 2, 1, 0, 7, 9, 4 };

    std::cout << "A: ";
    for (int i : A) {
        std::cout << i << ' ';
    }
    std::cout << '\n';

    // 오름차순 우선순위 큐
    std::priority_queue<int, std::vector<int>, std::greater<int>> pq(A, A + 10);

    pq.push(15);
    pq.push(-3);
    pq.push(13);
    pq.push(14);
    pq.push(-2);
    pq.push(11);
    pq.push(-4);
    pq.push(-5);
    pq.push(12);
    pq.push(-1);

    std::cout << "pq: ";
    while (!pq.empty()) {
        std::cout << pq.top() << ' ';
        pq.pop();
    }
    std::cout << '\n';
    return 0;
}
```

```
A: 3 6 5 8 2 1 0 7 9 4
pq: -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 11 12 13 14 15
```

Quiz

```
std::cout << "pq: ";
while (!pq.empty()) {
    std::cout << pq.top() << ' ';
    pq.pop();
}
std::cout << '\n';
```

pq.top을 통해 우선순위 큐를 출력하면 아래와 같은 결과가 나와야 함

짝수는 2로 나누었을 때 나머지가 0인 수, 홀수는 나머지가 0이 아닌 수

Hint

우선순위 큐에 대한 설명은 https://travelbeeee.tistory.com/126 를 참고할 것

비교 함수의 구현

```
bool cmp(int a, int b) {
    if (a % 2 == 0) {
        if (b % 2 == 0) {
            return a < b; // a짝 b짝
        }
        return true; // a짝 b홀
    }
    if (b % 2 == 0) {
        return false; // a홀 b짝
    }
    return a > b; // a홀 b홀
}
```

적절히 수정하여 사용할 것