인트아이 정연한

2022-2 인트아이 C++ 수업 자료

C++ 기초 3강

함수, 람다식, Class

CONTENTS

수학에서도 함수를 썼는데 여기서도?

프로그래밍 언어는 수학에서 많은 개념을 따온 만큼 수학과 관련 된 개념들이 매우 많다.
가장 대표적인 것이 바로 함수인데, 수학에서의 함수와 어떤 부분에서 다른지 알아보자.

```cpp
int lineFunc(int); // 함수 선언


int main() {
    int result = 0;
    // 결과값을 저장할 변수 선언
    result = lineFunc(10); // 함수 호출
    cout << result << endl; // 결과 출력
    return 0;
}

int lineFunc(int x) { // 함수 정의
    return 2 * x + 1;
}
```

수학에서도 함수를 썼는데 여기서도?

자, y = 2x + 1 이라는 일차함수가 있다고 치자.
우리는 이것을 C++ 함수로 바꿔줄 것인데, 우선 형태를 f(x) 형태로 바꿔주자.
y = f(x)를 위의 식에 대입해서
f(x) = 2x + 1 로 바꿔주었다.
일단 이 형태에서 조금씩만 바꿔주면 되는데, 변화 과정은 아래 사진을 참고하자

변화 과정을 하나 하나 살펴보니, C++의 함수도 수학에서의 함수와 크게 다르지 않다는게 보인다.

```
// y = 2x + 1을 프로그래밍 함수로 변환해보자
//
// y = 2x + 1                                <= y = f(x) 대입
//
// f(x) = 2x + 1                             <= 2와 x 사이에 생략 된 * 추가
//
// f(x) = 2*x + 1                            <= 함수 이름 f에서 lineFunc로 변경
//
// lineFunc(x) = 2*x + 1                     <= 매개변수에 자료형 추가
//
// lineFunc(int x) = 2*x + 1                 <= 함수의 반환형 추가
//
// int lineFunc(int x) = 2 * x + 1           <= 함수 본문 = 에서 중괄호로 변경
//
// int lineFunc(int x) { return 2 * x + 1 }
```

함수는 어떻게 써야 되지?

우선 함수는 사용을 하기 전 먼저 선언을 해주어야 한다.
선언 대신 정의를 바로 써주어도 되긴 하지만, 일반적으로는 선언과 정의를 분리해서 써준다.

함수 선언은 **해당 함수를 가져다 쓸 위치 위에** 써주어야 하며,
이 코드에서 lineFunc라는 함수를 가져다 쓰는(함수 호출) 위치는 main 함수이다.

코드를 잘 살펴보면, main 함수 안에서 lineFunc함수를 **호출** 한 뒤,
함수의 결과값을 result에 넣어주고 있다.

이때 lineFunc가 **호출** 된다 함은, lineFunc 함수를 **실행** 시킨다 라는 뜻과 같으며,
lineFunc를 실행시키면 매개변수 x로 들어온 10이라는 값에 의해
return 하는 식이 2 * 10 + 1로 치환되고, 이 식의 연산 결과인 **21이 return에 의해 반환**되게 된다.

이렇게 반환된 21은 반환값이라 불리며, 함수를 호출한 위치인 result = lineFunc(10) 위치에 들어가,
result = 21로 치환되게 된다.

그리고 그렇게 result의 값이 21로 바뀌어 21이 콘솔 화면에 출력되게 된다.

```cpp
int lineFunc(int); // 함수 선언


int main() {
    int result = 0;
    // 결과값을 저장할 변수 선언
    result = lineFunc(10); // 함수 호출
    cout << result << endl; // 결과 출력
    return 0;
}

int lineFunc(int x) { // 함수 정의
    return 2 * x + 1;
}
```

```
21

C:\Users\yt430\source\repos\2022-2-cpp-
었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

선언과 정의는 왜 분리 하는가

아까 잠시 설명했지만, 코드를 보면, 함수 선언을 먼저 하고, main함수에서 가져다가 쓰고,
main 함수 아래에서 lineFunc함수를 정의하는 것을 볼 수가 있는데,
여기서 알 수 있듯, C++에서는 함수의 선언과 정의를 분리 할 수 있다.

그렇다면 왜 함수의 선언과 정의를 분리해야 하는가?
답은 의외로 간단한데, **코드를 숨기기 위해**서이다.
예를 들어 내가 만든 함수가 꽤 이쁘게 잘 만들어졌다고 가정하자.
그래가지고 이 함수를 온라인에 배포를 하려 하는데,
나는 이 함수를 다른 사람이 썼으면 좋겠지만 함수를 어떻게 구현했는지(함수의 정의부분)
남들에게 보이고 싶지 않을 수 있다. 만약 유료로 판매되는 코드라면 특히나 더.

이런 상황에 나는 함수의 선언부, 즉 **함수 프로토타입 만을 공개**하고,
함수의 정의는 암호화 한 뒤 배포하여, **사람은 못읽고 컴퓨터만 읽을 수 있게** 배포 할 수도 있다.

결론적으로, 함수의 선언, 정의 분리는 **정보 은닉**을 위해 사용된다 라고 이해하면 된다.

물론 이 용도 말고도 그냥 코드를 좀 더 보기 좋게 하기 위해서도 분리하기도 한다.

```
int lineFunc(int); // 함수 선언


int main() {
    int result = 0;
    // 결과값을 저장할 변수 선언
    result = lineFunc(10); // 함수 호출
    cout << result << endl; // 결과 출력
    return 0;
}

int lineFunc(int x) { // 함수 정의
    return 2 * x + 1;
}
```

```
21

C:\Users\yt430\source\repos\2022-2-cpp-
었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

함수를 좀 더 자세히 알아보자

함수를 어떻게 쓰는지에 대해서는 대충 알았으니, 함수의 구조에 대해 좀 더 자세히 알아보자.
함수는 기본적으로 **반환형 함수이름(매개변수)** 의 구조로 이뤄진다.

반환형이란 이 함수가 반환하는 값의 자료형을 의미한다.
아까전 우리는 return을 통해서 함수의 결과값을 반환할 수 있다는 것을 알았는데,
이 결과값의 자료형이 바로 반환형인 것이다.

당연하지만 return문이 오면 함수는 값을 반환하면서 그대로 종료가 되게 된다.

함수 이름은 그냥 함수의 식별자이고,
매개변수는 함수를 호출하면서 함수 안에 값을 넣어주기 위한 장치이다.

함수를 호출 할 때, 호출하는 위치에서는 **함수이름(인자)**를 통해서 함수를 호출하게 되는데,
이렇게 들어온 인자값을 매개변수가 받아서 호출 된 함수 내부에서 사용할 수 있다.

당장 이 코드에서도 10을 인자값으로 넘겨서 x라는 매개변수가 그 값을 받아,
반환식에서 x를 10으로 치환하여 2 * 10 + 1을 만들어 21이란 값을 반환하는데 사용되었다.

```cpp
int lineFunc(int); // 함수 선언


int main() {
    int result = 0;
    // 결과값을 저장할 변수 선언
    result = lineFunc(10); // 함수 호출
    cout << result << endl; // 결과 출력
    return 0;
}

int lineFunc(int x) { // 함수 정의
    return 2 * x + 1;
}
```

```
21

C:\Users\yt430\source\repos\2022-2-cpp-
었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

사실 반환 값과 매개변수는 없을 수 있다

방금 전까지 우리는 수학에서의 일차 함수를 그대로 프로그래밍 함수로 옮겨 사용해보았다.
이번에는 수학에서의 함수와 다른 C++만의 함수를 확인해보자.

사실 C++에서는 함수의 **반환 값이나 매개변수가 없거나 둘 다 없을** 수 있다.
첫 번째 함수인 func1은 반환 값이 존재하고, 매개변수 역시 존재하지만,
두 번째 함수인 func2는 반환 값이 존재하지 않고, 매개변수 역시 존재하지 않는다.

그런데 실제로 func2 함수를 호출해보면 잘만 실행 되는 것을 볼 수 있다.
이런식으로 함수의 반환값과 관계 없이 **함수 외부에 영향을 미치는 것**을
부작용(side effect)라고 부른다.

이 코드에서 func2함수는 반환값이 없으며, 외부에 문자열 출력만을 하는 **부작용만 존재하는 함수**라 볼 수 있을 것이다.

반대로 **부작용이 없으면서**, 언제 해당 함수를 호출해도 **항상 같은 값을 반환** 하는 함수를 **순수 함수**라 부른다.

이 코드에서 func1이 순수함수의 한 예로 볼 수 있을 것이다.
a에 10을 집어 넣으면 언제 10을 집어넣든 항상 20이 반환될 것이고, 부작용도 존재하지 않기 때문.

```cpp
int func1(int a) {
    return a * 2;
}

void func2() {
    cout << "부작용만 존재하는 함수" << endl;
}

int main() {
    int i = 5;
    cout << func1(i) << endl;
    func2();
    return 0;
}
```

```
10
부작용만 존재하는 함수
```

반환형이 void라는건 반환을 못하는 것일까?

정답은 **그렇지 않다** 이다. 반환형이 void라 할지라도 **return 구문을 사용할 수는 있다**.
단, return 뒤에 어떤 **값이 올 수는 없고**, 따라서 **반환값이 존재할 수 없다**.

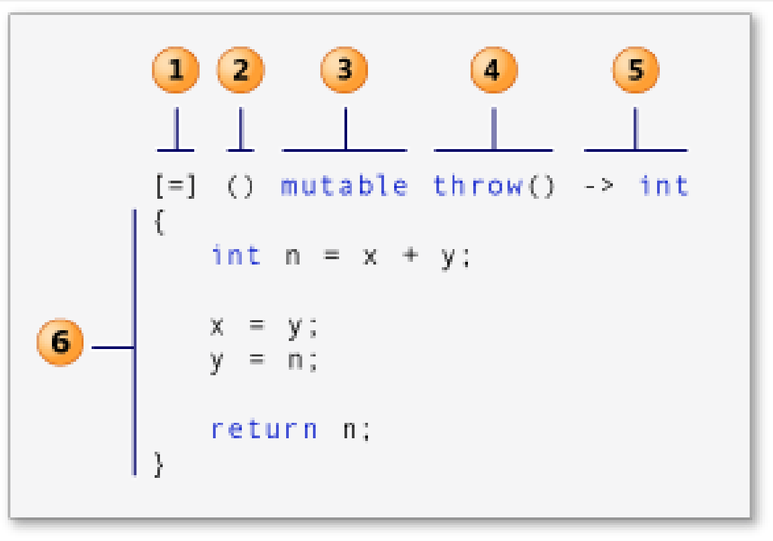
따라서 아래쪽 main함수에서 int i에다가 func함수의 반환값을 집어넣으려는 짓은 불가능하다.

사실 좀 더 엄밀히 따지자면 void형 데이터가 존재한다면 반환을 할 수도 있겠지만,
그런 데이터는 존재하지 않기 때문에 반환값이 존재할 수가 없다.

```
void func() {
    return 0; // 불가능
    return; // 함수를 끝내는 용도로만 기능
}

int main() {
    int i = func(); // 반환값이 없으니 불가능
    return 0;
}
```

람다식? 익명 함수?



사실 방금 전에 막 함수를 배운 참에 람다식을 배우는 것이 옳은가에 대한 의문점이 있긴 하지만,
요즘 워낙에 함수형 프로그래밍이 인기이기에 람다식도 진행하고자 한다.

우선 람다식이란 자세히 파고들면 매우 복잡한 개념이기에 프로그래밍에서 자주 사용되는 개념으로 설명하자면,
함수를 더 단순하게 표현하여 **함수의 계산을 좀 더 용이**하게 하는 일종의 표기법(또는 계산 방법)이다.
쉽게 보자면 그냥 **함수를 간단하게 표현한 것** 이라고 볼 수도 있지만, C++에서는 **일반 함수와는 조금 다르게 동작**한다.

기본적으로 C++에선 함수 안에 함수를 정의하는 것은 불가능하다. 함수 선언이라면 모를까
하지만 람다식을 사용하면 함수 안에 함수를 정의하는 짓이 가능해진다.

또한 일반 함수는 매개변수의 자료형으로 auto를 쓸 수 없으나, 람다식은 가능하다. 이를 통해 좀 더 유연한 프로그래밍이 가능하다.

이것 말고도 함수의 인자로 람다식을 넘길 수 있다든지, 함수의 반환값으로 람다식을 반환 할 수 있다든지, 이런 다양한 것들이 가능하고,
또, 이렇게 많이 쓰인다.

람다식의 다른 이름으로 람다 대수, 람다 함수, 익명 함수 등이 있는데, 이 중 익명 함수는 람다식을 통해 익명함수를 구현할 수 있기에,
람다식을 그냥 익명 함수라고 부르기도 한다.

간단한 예제

람다식에 대한 자세한 이해는 공식 문서를 참고하는게 좋고, 여기서는 간단한 사용례만 소개하고자 한다.
(공식 문서 링크: https://docs.microsoft.com/ko-kr/cpp/cpp/lambda-expressions-in-cpp?view=msvc-170)

보통 일반적으로 람다식에는 화살표가 붙기 때문에 다른 언어에선 **화살표 함수**라고도 부르는데,
C++에서는 화살표를 생략하고 쓸 수 있다.

아래의 코드를 보면, y1과 y2에 각각 똑같은 람다식이 들어간 것이 보인다.
차이점은 화살표를 생략했냐 안했냐 차이.

그리고 y1과 y2를 auto로 선언했는데, 특별한 경우가 아니라면, 일반적으로 람다식을 받는 변수는 auto로 선언한다.

y1과 y2로 함수를 받는게 잘 이해가 안된다면, 수학에서 **y = f(x)** 와 같다고 보면 된다.
이 경우는 **y = 2x + 1를 조금 다르게 표현 한 것**이라 보면 된다.

이렇게 받은 함수는 **일반 함수를 쓸 때 처럼 y(인자)** 이런식으로 쓸 수 있다.

실제로 이런식으로 함수를 호출하고 결과를 보면 정상적으로 21이 출력 된 것을 볼 수 있다.

```cpp
int main() {
    // 화살표 생략 안한 경우
    auto y1 = [](int x) -> int { return x * 2 + 1; };

    // 화살표 생략 한 경우
    auto y2 = [](int x) { return x * 2 + 1; };

    int result1 = y1(10);
    int result2 = y2(10);

    cout << result1 << endl;
    cout << result2 << endl;

    return 0;
}
```

```
21
21

C:\Users\yt430\source\repos\2022-2-cpp-
었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

IIFE(정의와 동시에 실행되는 함수)

IIFE는 Immediately Invoked Function Expression의 약자로 즉시 호출된 함수식 이라는 뜻인데,
람다식은 함수를 정의하자마자 바로 실행을 시킬 수 있다.
아래에 있는 코드를 보면 알 수 있듯, 람다식 끝에 아까는 없던 소괄호와 인자값이 들어가 있는 것을 볼 수 있다.
이는 [] () {} 로 이뤄진 람다식을 다른 변수에 할당하지 않고 그대로 실행을 시킨 것인데,
이는
auto y = [](int x) { return x* 2 + 1; };
cout << y(10) << endl;
을 한 것과 같다.

```cpp
int main() {
    cout << [](int x) { return x * 2 + 1; }(10) << endl;

    return 0;
}
```

```
21

C:\Users\yt430\source\repos\2022-2-cpp-
었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

Class가 뭐지...?

영어 단어 Class는 종류, 계급, 계층 등의 뜻을 가지고 있다.
그 뜻 처럼, C++에서 클래스는 **같은, 또는 비슷한 속성을 가진 개념들을 어느 하나로 정의 내린 것**. 이라고 볼 수 있다.
예를 들어 동물이라는 클래스가 있다고 치자.
이 **동물 이라는 클래스**에는 **토끼**도 해당할 수 있고, **강아지**도 해당할 수 있고, **무당벌레**도 해당할 수 있고, **복어**도 해당할 수 있을것이다.
이처럼 **같은 속성을 가진 개념**들은 **하나의 클래스로 정의** 내릴 수 있다.
따라서 아래의 그림에서 **종, 속, 과, 목, 강, 문, 계**는 **각각이 전부 클래스**에 해당한다고 볼 수 있다!
물론 아래같은 특수한 경우 종에 해당하는 클래스가 계에 해당하는 클래스보단 훨씬 더 작은 집합이겠지만, 일단은 신경쓰지 말자.

클래스의 예시

클래스가 뭔지는 대충은 알겠는데 그렇다면 이제 예시를 하나 보면서 C++에서의 클래스를 좀 더 자세히 알아보자.
C++에서 클래스는 다음 그림처럼 선언할 수 있다.
class 클래스의 이름 {
public:

private:

}
여기서는 밀키트 라는 이름으로 클래스를 선언했다.

클래스는 기본적으로 두가지 부분으로 구분되는데, 하나는 public 영역, 나머지 하나는 private 영역이다.
public 영역은 **해당 클래스와 관련된 기능**들을 **멤버 함수**를 통해 정의하는 부분이고,
private 영역은 **해당 클래스를 이루는 여러가지 속성**들을 **멤버 변수**를 통해 정의하는 부분이다.

```
class 밀키트 {
public: // 밀키트와 관련된 기능(function)을 멤버 함수로 정의한다.

private: // 밀키트라는 개념을 이루는 여러 속성을 멤버 변수로 정의한다.

};
```

멤버 함수? 멤버 변수?

멤버 함수란
클래스 내에 속하는 함수로, **해당 클래스와 관련된 기능**이 들어가야 한다.
아래는 밀키트 클래스인데, 그렇다면 해당 클래스에 들어갈 기능은 **요리와 관련된 기능**이어야지,
갑자기 **운동과 관련 된 기능**이 들어가면 **안된다**는 것이다.
참고로 멤버 함수는 **메서드** 라고도 불린다.

멤버 변수란
클래스 내에 속하는 변수로 **해당 클래스의 개념을 구성하는 여러가지 속성**들이 멤버 변수에 해당한다.
위와 마찬가지로 밀키트 클래스에는 **요리의 이름**이나 **조리 온도** 같은 것들이 들어가야지,
갑자기 **다리 개수**라든가, **전압** 같은 전혀 상관 없는 개념이 들어가면 **안된다**는 것이다.
참고로 멤버 변수는 **필드** 라고도 불린다.

```
class 밀키트 {
public: // 밀키트와 관련된 기능(function)을 멤버 함수로 정의한다.

private: // 밀키트라는 개념을 이루는 여러 속성을 멤버 변수로 정의한다.

};
```

멤버 변수부터 정의해보자

아까 전 멤버 변수는 **해당 클래스의 개념을 구성하는 여러 속성들**이라고 했다.
자, 그렇다면, 이 밀키트 클래스에는 해당 밀키트의 이름, 해당 밀키트의 종류, 조리하는데 걸리는 시간, 조리하는 온도 등을 쓸 수 있을것이다.

더 많은 속성들이 존재하겠지만, 일단은 요 정도만 정의해두자.

자 그럼 각각의 속성들을 일단 변수로 선언해보자.
이름은 문자열일테니 **string**으로 정의할 수 있겠다.
종류는 구현 방법에 따라 정수로도 정의할 수 있겠지만 나는 문자열, 즉 **string**으로 정의하겠다.
조리 시간은 정수값으로 나올테니 **int**로 정의하고,
조리 온도는 36.5도처럼 실수값이 나올 수 있으니 **double**로 정의해주자.

```
private: // 밀키트라는 개념을 이루는 여러 속성을 멤버 변수로 정의한다.
    string 이름; // 해당 밀키트의 이름
    string 종류; // 해당 밀키트의 종류
    int 조리_시간; // 해당 밀키트를 조리하는데 걸리는 시간
    double 조리_온도; // 해당 밀키트를 조리하는 온도
};
```

멤버 함수도 정의해보자

멤버 함수는 해당 클래스와 관련된 기능들이 들어간다고 했다.
자 그럼 밀키트와 관련된 기능... 이라 하면 조리 방법(cooking method)가 들어갈 수 있을 것이다.
근데 아래 코드를 보니 조리 방법은 없고 웬 **클래스의 이름과 똑같은 이름을 가진 함수**가 하나 있다.

아니... 그 전에 함수는 맞나? 반환형이 없는데...?

결론부터 말하자면 함수가 맞긴 맞다. 근데 좀 **특수한 함수**인데,
이렇게 **클래스의 이름을 함수의 이름으로 하는** 특수한 함수를 우리는 **생성자** 라고 부른다.

```
class 밀키트 {
public: // 밀키트와 관련된 기능(function)을 멤버 함수로 정의한다.
    밀키트(string 이름, string 종류, int 조리_시간, double 조리_온도) {
        this->이름 = 이름;
        this->종류 = 종류;
        this->조리_시간 = 조리_시간;
        this->조리_온도 = 조리_온도;
    }
```

생성자...?가 뭔데?

생성자는 클래스에 존재하는 특수한 함수로, 이름 그대로 무언가를 생성하는 역할을 한다.
클래스는 하나의 틀과 같은데, 이 틀을 통해서 **객체** 라는것을 찍어 낼 수 있다.
이때, 이 **객체**를 **찍어내면서** 각각의 객체에 **고유한 값**을 **넣어**주는게 바로 **생성자**의 역할이다.

생성자의 엄밀한 정의는, **객체가 생성될 때 호출되어, 객체를 초기화 하는 역할**을 하는 함수인데,

이렇게 각각의 객체에 고유한 값을 넣어주는 것은, 즉 초기화를 의미한다고 볼 수 있다.
이전에도 배웠듯이, **초기화란 변수를 선언할 때, 초기값을 넣어주는 것**이라 했으니까!

```
class 밀키트 {
public: // 밀키트와 관련된 기능(function)을 멤버 함수로 정의한다.
    밀키트(string 이름, string 종류, int 조리_시간, double 조리_온도) {
        this->이름 = 이름;
        this->종류 = 종류;
        this->조리_시간 = 조리_시간;
        this->조리_온도 = 조리_온도;
    }
```

생성자 말고 다른 멤버 함수는?

아까전에 말했듯, 밀키트와 관련된 기능이라 하면, 조리법이 해당 할 수 있다고 했었다.
아래는 간단한 조리법을 출력하는 기능을 하는 멤버 함수 조리법 을 정의해두었는데,
생성자를 통해 초기화된 객체가 어떤 값을 가지느냐에 따라서 다른 조리법이 출력 될 것이다!

```
void 조리법() {
    cout << 이름 << "을(를) 조리하는 방법:" << endl;

    // 입력받은 멤버 변수 {종류} 값에 따라 조리법이 달라진다.
    if (종류 == "볶음") {
        cout << "밀키트 내에 있는 재료를 다 꺼내서 메인 재료와 채소를 다듬어 주고," << endl
            << "궁중 팬이나 웍을 " << 조리_온도 << "도로 가열하여 메인 재료와 채소들을 넣고, " << endl
            << 조리_시간 << "분 만큼 볶다가 향신료를 뿌려준다." << endl;
    }
    else if (종류 == "탕") {
        cout << "밀키트 내에 있는 육수를 꺼내 냄비에 넣고," << endl
            << "물을 약간 넣어 어느정도 양을 맞추고," << endl
            << 조리_온도 << "도로 가열시켜, 나머지 재료들을 몽땅 때려 넣고," << endl
            << 조리_시간 << "분간 익힌다." << endl;
    }
    else if (종류 == "구이") {
        cout << "밀키트 내에 있는 메인 재료를 꺼내 향신료에 마리네이드 한 후," << endl
            << "팬을 꺼내 기름을 두르고, " << 조리_온도 << "도로 달궈준 다음," << endl
            << 조리_시간 << "분간 메인 재료를 기름에 튀기듯이 구워준다." << endl;
    }
}
```

이렇게 정의는 했는데, 어떻게 써?

자, 이렇게 간단한 클래스 하나를 정의내렸다.
이제 이 클래스를 사용해 객체를 만들어볼 차례!
객체는 아래와 같이 변수를 선언하는 것처럼 클래스의 이름을 앞에 쓰고, 그 뒤에 객체의 이름,
그 뒤에 해당 클래스의 생성자에 들어갈 인자값들을 넣어서 만들 수 있다!

```
int main() {
    // 위에서 만든 클래스를 통해 객체를 선언할 수 있다.
    밀키트 감자탕{ "감자탕", "탕", 10, 100 };

    감자탕.조리법();


    밀키트 스테이크{ "스테이크", "구이", 6, 170 };
    스테이크.조리법();

    밀키트 제육볶음{ "제육볶음", "볶음", 12, 160 };
    제육볶음.조리법();

    return 0;
}
```

이렇게 정의는 했는데, 어떻게 써?

지금 보면 3개의 객체를 선언하고 있는데, 각각의 객체들은 전부 밀키트 라는 클래스를 통해 마치 변수처럼 선언을 하고 있다.
따라서 클래스를 사용자 정의 자료형(User-Defined Types)라고도 부른다!
그리고 밀키트 객체 감자탕을 보면 감자탕.조리법(); 이런식으로 우리가 정의한 조리법 멤버함수를 호출하고 있는걸 볼 수가 있다!
이처럼 객체에서 해당 객체가 갖고 있는 특정한 값이나 함수를 호출하기 위해선 dot 연산자(.)를 사용해 호출 할 수 있다.

```
int main() {
    // 위에서 만든 클래스를 통해 객체를 선언할 수 있다.
    밀키트 감자탕{ "감자탕", "탕", 10, 100 };

    감자탕.조리법();


    밀키트 스테이크{ "스테이크", "구이", 6, 170 };
    스테이크.조리법();

    밀키트 제육볶음{ "제육볶음", "볶음", 12, 160 };
    제육볶음.조리법();

    return 0;
}
```

실행 결과는?

각각의 조리법들이 전부 잘 호출 된 것이 보인다!!

```
int main() {
    // 위에서 만든 클래스를 통해 객체를 선언할 수 있다.
    밀키트 감자탕{ "감자탕", "탕", 10, 100 };

    감자탕.조리법();


    밀키트 스테이크{ "스테이크", "구이", 6, 170 };
    스테이크.조리법();

    밀키트 제육볶음{ "제육볶음", "볶음", 12, 160 };
    제육볶음.조리법();

    return 0;
}
```

```
감자탕을(를) 조리하는 방법:
밀키트 내에 있는 육수를 꺼내 냄비에 넣고,
물을 약간 넣어 어느정도 양을 맞추고,
100도로 가열시켜, 나머지 재료들을 몽땅 때려 넣고,
10분간 익힌다.
스테이크을(를) 조리하는 방법:
밀키트 내에 있는 메인 재료를 꺼내 향신료에 마리네이드 한 후,
팬을 꺼내 기름을 두르고, 170도로 달궈준 다음,
6분간 메인 재료를 기름에 튀기듯이 구워준다.
제육볶음을(를) 조리하는 방법:
밀키트 내에 있는 재료를 다 꺼내서 메인 재료와 채소를 다듬어 주고,
궁중 팬이나 웍을 160도로 가열하여 메인 재료와 채소들을 넣고,
12분 만큼 볶다가 향신료를 뿌려준다.
```

과제 1. 백준 3003번 킹, 퀸, 룩, 비숍, 나이트, 폰

```cpp
constexpr int NumberOfMember = 6;

// 반환값이 없고, int 배열을 인자로 받는 함수
// 배열에 대한 설명은 여기 참조
// https://boycoding.tistory.com/193
void calculator(int[NumberOfMember]);

int main() {
    int chessPieces[NumberOfMember] = { };
    for (int i = 0; i < NumberOfMember; ++i)
        cin >> chessPieces[i];

    calculator(chessPieces);

    return 0;
}

void calculator(int chessPieces[NumberOfMember]) {
    // TODO: 여기에 문제 풀이 작성
}
```

문제

동혁이는 오래된 창고를 뒤지다가 낡은 체스판과 피스를 발견했다.
체스판의 먼지를 털어내고 걸레로 닦으니 그럭저럭 쓸만한 체스판이 되었다. 하지만, 검정색 피스는 모두 있었으나,
흰색 피스는 개수가 올바르지 않았다.
체스는 총 16개의 피스를 사용하며, 킹 1개, 퀸 1개, 룩 2개, 비숍 2개, 나이트 2개, 폰 8개로 구성되어 있다.
동혁이가 발견한 흰색 피스의 개수가 주어졌을 때, 몇 개를 더하거나 빼야 올바른 세트가 되는지 구하는 프로그램을 작성하시오.
단, 오른쪽의 코드를 참고하여 반드시 함수 또는 람다식을 사용하여 문제를 풀 것

입력

첫째 줄에 동혁이가 찾은 흰색 킹, 퀸, 룩, 비숍, 나이트, 폰의 개수가 주어진다. 이 값은 0보다 크거나 같고 10보다 작거나 같은 정수이다.

출력

첫째 줄에 입력에서 주어진 순서대로 몇 개의 피스를 더하거나 빼야 되는지를 출력한다.
만약 수가 양수라면 동혁이는 그 개수 만큼 피스를 더해야 하는 것이고, 음수라면 제거해야 하는 것이다.

예제 입력 1 복사	예제 출력 1 복사
0 1 2 2 2 7	1 0 0 0 0 1
예제 입력 2 복사	예제 출력 2 복사
2 1 2 1 2 1	-1 0 0 1 0 7

과제 2. 백준 27294번 몇개고?

문제

동우는 주영이가 '재벌집 막내아들'을 보게 하기 위해 인트아이 사람들을 모았다. 주영이에게 그냥 영상을 보여주는 것보다 알고리즘을 이용해서 알려주어야 흥미를 가지게 할 수 있다고 생각한 인트아이 사람들은 주영이가 드라마에 흥미를 가지게 하기 초밥 밥알 갯수로 문제를 만들기로 했다. 자 인트아이에서 문제를 만들었고, 동우가 주영이에게 문제를 말했다.

동우: "오늘 나는 너가 만든 초밥을 먹을 거야. 너는 '재벌집 막내아들'의 진양철 회장님의 말에 따라 술하고 같이 초밥을 먹거나 점심 식사가 아닐 때는 초밥의 밥알을 280개로 하며, 점심 식사이면서 술과 같이 먹지 않을때는 초밥의 밥알을 320개로 하여 초밥을 만들어야 해, 근데, 내가 초밥을 언제 먹을지, 혹은 술과 같이 먹을지 아직 정하지 않았어. 내가 초밥을 먹는 시간과 술의 유무를 말하면, 그때 너는 너가 만들어야 하는 초밥의 밥알 갯수를 구해야해"

당신은 위의 문제를 읽고 주영이를 대신하여 동우의 문제를 해결하여라.

단, 반드시 초밥_장인 클래스를 구현하여 해당 문제를 풀 것

입력

첫 번째 줄에 시간을 의미하는 정수

T(0 <= T <= 23)와 술의 유무를 의미하는 정수, S(0 <= S <= 1)가 공백으로 구분되어 주어진다.

T가 11 이하이면 아침 시간, 12 이상 16 이하이면 점심 시간, 그렇지 않으면 저녁 시간을 뜻한다.

S가 1 이면 술과 함께 먹는 것을 의미하며, 0 이면 술 없이 초밥을 먹는 것을 의미한다.

출력

첫 번째 줄에 주영이가 만들어야 하는 초밥 하나의 밥알의 갯수를 출력하여라.

예제 입력 1 복사	예제 출력 1 복사
16 0	320

점심 식사이며 술이 없으므로 320개이다.

예제 입력 2 복사	예제 출력 2 복사
14 1	280

점심 식사지만 술이 있으므로 280개이다.

예제 입력 3 복사	예제 출력 3 복사
17 0	280

저녁 식사 이므로 280개이다.

예제 입력 4 복사	예제 출력 4 복사
10 1	280

예제 입력 5 복사	예제 출력 5 복사
10 0	280