

2022-2 인트아이 C++ 수업 자료

C++ 기초 4강 함수, 람다식





CONTENTS

01. 함수

02. 람다식

03. 오늘의 과제

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.

```
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.

// $y = 2x + 1$ 을 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
//	
// $\text{lineFunc}(x) = 2*x + 1$	<= 매개변수에 자료형 추가
//	
// $\text{lineFunc}(\text{int } x) = 2*x + 1$	<= 함수의 반환형 추가
//	
// $\text{int lineFunc}(\text{int } x) = 2 * x + 1$	<= 함수 본문 = 에서 중괄호로 변경
//	
// $\text{int lineFunc}(\text{int } x) \{ \text{return } 2 * x + 1 \}$	

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 $2 * 10 + 1$ 로 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 = 21`로 치환되게 된다.

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

```
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 위해서**이다.

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 공개하고,

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 $2 * 10 + 1$ 을 만들어 21이란 값을 반환하는데 사용되었다.

```
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 것을**
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.

```
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.

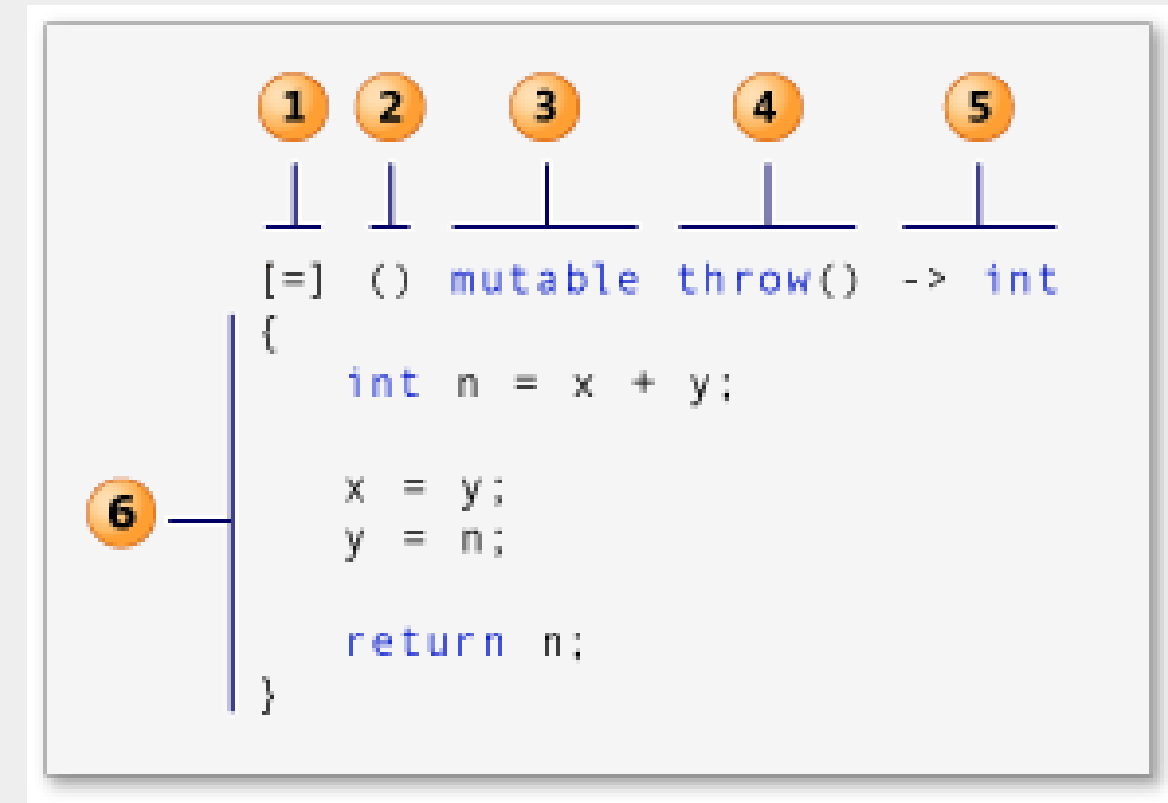
우선 람다식이란 자세히 파고들면 매우 복잡한 개념이기에 프로그래밍에서 자주 사용되는 개념으로 설명하자면, **함수를 더 단순하게 표현하여 함수의 계산을 좀 더 용이하게 하는 일종의 표기법(또는 계산 방법)이다.** 쉽게 보자면 그냥 **함수를 간단하게 표현한 것** 이라고 볼 수도 있지만, C++에서는 **일반 함수와는 조금 다르게 동작한다.**

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 $y = 2x + 1$ 를 조금 다르게 표현 한 것이라 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.

```
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)

마지막으로 IIFE만 소개하고 마치도록 하겠다.

IIFE는 Immediately Invoked Function Expression의 약자로 즉시 호출된 함수식이라는 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.

```
int main() {  
    auto y = [](int x) { return x * 2 + 1; }(10);  
  
    cout << y << endl;  
  
    return 0;  
}
```

21

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

03 오늘의 과제

백준 3003번 킹, 퀸, 룯, 비숍, 나이트, 폰

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 올바르게 않았다.

체스는 총 16개의 피스를 사용하며, 킹 1개, 퀸 1개, 룯 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 함수를 사용하여 문제를 풀 것

입력

첫째 줄에 동혁이가 찾은 흰색 킹, 퀸, 룯, 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.

```
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
}
```

예제 입력 1 복사

0 1 2 2 2 7

예제 입력 2 복사

2 1 2 1 2 1

예제 출력 1 복사

1 0 0 0 0 1

예제 출력 2 복사

-1 0 0 1 0 7