

콜백

프로그래밍에서 **콜백(callback)** 또는 **콜백 함수(callback function)**는 다른 코드의 인수로서 넘겨주는 실행 가능한 코드를 말한다. 콜백을 넘겨받는 코드는 이 콜백을 필요에 따라 즉시 실행할 수도 있고, 아니면 나중에 실행할 수도 있다.



일반적으로 콜백수신 코드로 콜백 코드(함수)를 전달할 때는 콜백 함수의 **포인터 (핸들)**, **서브루틴** 또는 **람다함수**의 형태로 넘겨준다. 콜백수신 코드는 실행하는 동안에 넘겨받은 콜백 코드를 필요에 따라 호출하고 다른 작업을 실행하는 경우도 있다. 다른 방식으로는 콜백수신 코드는 넘겨받은 콜백 함수를 '핸들러'로서 등록하고, 콜백수신 함수의 동작 중 어떠한 반응의 일부로서 나중에 호출할 때 사용할 수도 있다 (비동기 콜백). 콜백은 **폴리모피즘**과 **제네릭프로그래밍**의 단순화된 대체 수법이며, 콜백 수신 함수의 정확한 동작은 콜백 함수에 의해 바뀐다. 콜백은 **코드 재사용**을 할 때 유용하다.

배경

콜백을 사용하는 의의를 이해할 수 있는 한 예로 **연결 리스트** 상의 각 요소에 대해서 여러 가지 처리를 수행하는 문제를 생각해볼 수 있다. 한 방법으로서 리스트 상의 **반복자(iterator)**로 각 객체를 처리하게 할 수 있다. 이것은 실제로 가장 일반적인 방법이지만, 이상적인 방법은 아니다. 반복자를 제어하는 코드(예를 들면 **for** 문)는 리스트의 노드를 방문할 때마다 노드를 복제해야 한다. 게다가 리스트의 갱신이 비동기 프로세스로 처리되는 경우, 반복자로 리스트를 탐색하는 동안에 요소를 잃어버리거나 다음 노드를 탐색할 수 없게 될 가능성이 있다.

대체 방법으로 새롭게 라이브러리 함수를 만들어, 적당한 동기신호를 통해 필요한 처리를 하도록 한다. 이 경우에도 리스트를 탐색할 때마다 동일한 함수를 호출해야 한다. 이 방식은 여러 응용 프로그램에 쓰이는 범용 라이브러리에는 적합하지 않다. 라이브러리를 개발할 때에는 모든 응용 프로그램의 요구를 예측할 필요가 없게 하며, 응용 프로그램 개발에서는 라이브러리에 추가된 기능에 대한 자세한 정보를 알 필요가 없도록 하는 것이 바람직하다.

콜백을 사용하면 이러한 문제를 해결할 수 있다. 리스트를 탐색하고자 할 때, 프로그래머는 응용 프로그램이 각 요소를 처리하는 방법을 콜백 코드로 제공한다. 이런 식으로 유연성을 해치지 않고 명확하게 라이브러리와 응용 프로그램을 구별할 수 있다.

예시

아래의 **C** 코드는 배열을 탐색하여 5 보다 큰 값을 찾는 일을 한다. 먼저 반복자를 사용한 코드를 살펴보자.

```
int i;
for (i = 0; i < length; i++) {
    if (array[i] > 5) {
        break;
    }
}
```

```

if (i < length) {
    printf("Item %d\n", i);
} else {
    printf("Not found\n");
}

```

다음으로, callback이라는 함수 포인터를 이용해 콜백을 구현한 코드를 살펴보자.

```

/* 라이브러리 코드 */
int traverseWith(int array[], size_t length,
                int (*callback)(int index, int item, void *param),
                void *param)
{
    int exitCode = 0;
    for (int i = 0; i < length; i++) {
        exitCode = callback(i, array[i], param);
        if (exitCode) {
            break;
        }
    }
    return exitCode;
}

/* 응용 프로그램 코드 */
int search (int index, int item, void *param)
{
    if (item > 5) {
        *(int *) param = item;
        return 1;
    } else {
        return 0;
    }
}

/* 라이브러리를 호출하는 본체 */
int index;
int found;
found = traverseWith(array, length, search, &index);
if (found) {
    printf("Item %d\n", index);
} else {
    printf("Not found\n");
}

```

콜백 함수 **search**의 if 문의 조건을 변경하면, 「5보다 크다」 이외의 요소를 검색하는 데에도 사용할 수 있다. **traverseWith**에는 콜백이 자신의 목적을 위해서 받는 추가의 인수 **param**이 있는 점에 주목할 필요가 있다. 일반적인 콜백의 경우 이러한 인수를 변수 영역 외의 응용 프로그램 데이터 포인터에 이용한다. 이는 정적 영역 방식의 언어(C 나 C++)에만 필요하다 (다만, C++를 포함한 객체 지향 언어에는 다른 해결책이 있다). 동적 영역 언어(일부 함수형 언어 등)의 경우 클로저를 이용하면 자동으로 응용 프로그램 데이터로 접근할 수 있다. 예를 들어, 같은 프로그램을 LISP로 쓰는 경우를 살펴보자.

```

; 라이브러리 코드
(defun traverseWith (array callback)
  (let ((exitCode nil)
        (i 0))
    (while (and (not exitCode) (< i (length array)))
      (setq exitCode (callback i (aref array i)))
      (setq i (+ i 1)))
    exitCode))

; 응용 프로그램 코드
(let (index found)
  (setq found (traverseWith array (lambda (idx item)
                                     (if (<= item 5) nil
                                         (setq index idx)
                                         t))))))

```

이 경우 콜백 함수는 사용하는 시점에서 정의되어 "**index**"를 이름으로 참조하고 있다. 이 예에서는 동기화를 고려하지 않았지만, **traverseWith** 함수를 동기화할 수 있도록 대처하는 것이 더 쉽다. 게다가 더 중요한 것은 그 함수를 수정하는 것만으로도 동기화 여부를 결정할 수 있다는 점이다.

같이 보기

- [async/await](#)

각주

외부 링크

- [Style Case Study #2: Generic Callbacks \(http://gotw.ca/gotw/083.htm\)](http://gotw.ca/gotw/083.htm)

원본 주소 "<https://ko.wikipedia.org/w/index.php?title=콜백&oldid=33268284>"

▪