과 목 명

C언어I

함수 사용 전·후 프로그램 비교와

함수화의 장점

과 제 보 고 서

제출자 : _____ 제출일 : _____

목 차

I. 서론

C언어 프로그램은 모든 명령을 main 함수 안에 순서대로 작성하는 방식으로도 만들 수 있다. 규모가 매우 작은 예제에서는 이러한 방식이 간단해 보일 수 있다. 그러나 프로그램이 길어지고 같은 계산이나 처리 과정이 여러 번 필요해지면 코드가 반복되고, 어디에서 오류가 발생했는지 찾기 어려워진다. 이때 기능별로 코드를 함수로 나누면 프로그램을 여러 개의 작은 단위로 설계할 수 있어 구조가 명확해진다.

함수는 특정 작업을 수행하는 명령문의 묶음으로, 필요한 입력값을 매개변수로 받고 결과를 반환값으로 돌려줄 수 있다. 표준 라이브러리의 printf, scanf, strlen 등도 모두 미리 만들어진 함수를 호출하여 사용하는 예이다. 사용자가 직접 함수를 정의하면 자신이 자주 사용하는 계산이나 처리도 반복해서 활용할 수 있다. 본 과제에서는 함수를 사용하지 않은 프로그램과 함수를 활용한 프로그램을 비교하고, 함수화가 제공하는 장점을 학생 점수 처리 예시를 통해 설명하고자 한다.

II. 함수의 개념과 필요성

1. 함수의 역할

C언어에서 함수는 "하나의 목적을 수행하는 독립된 코드 블록"으로 볼 수 있다. 함수는 반환형, 함수 이름, 매개변수 목록, 함수 본문으로 구성되며, 다른 함수에서 이름을 통해 호출할 수 있다. 예를 들어 두 정수의 합을 구하는 기능을 add 함수로 정의하면 main에서는 add(3, 5)처럼 호출할 수 있다. 중요한 점은 호출하는 쪽이 함수 내부의 모든 구현을 알지 못해도 입력과 출력 규칙만 알면 기능을 사용할 수 있다는 것이다. 이러한 특성은 복잡한 프로그램을 계층적으로 설계하게 해 준다.

2. 함수 없이 작성한 프로그램의 문제

모든 코드를 main 안에 작성하면 처음에는 흐름을 한눈에 볼 수 있지만, 코드가 길어질수록 한 함수가 입력, 계산, 출력, 예외처리까지 모두 담당하게 된다. 같은 계산이 여러 곳에서 필요하면 동일

코드를 복사하여 붙여 넣게 되고, 계산식을 수정할 때 복사된 모든 부분을 찾아 바꾸어야 한다. 한 곳을 빠뜨리면 서로 다른 결과가 발생할 수도 있다. 또한 오류가 생겼을 때 문제가 입력 부분인지 계산 부분인지 출력 부분인지 경계를 나누기 어려워 디버깅 시간이 길어진다.

III. 함수 사용의 주요 장점

1. 모듈화와 가독성

첫 번째 장점은 모듈화이다. 하나의 큰 문제를 입력, 계산, 검색, 출력처럼 작은 기능으로 분해하고 각 기능을 함수로 만들면 프로그램의 전체 구조를 쉽게 이해할 수 있다. main 함수에는 "어떤 순서로 작업이 진행되는가"만 남고 세부 구현은 개별 함수가 담당한다. 함수 이름을 calculate_average, find_max처럼 목적이 드러나게 정하면 긴 반복문이나 계산식을 직접 읽지 않아도 프로그램의 의도를 파악할 수 있다.

2. 재사용성과 유지보수

두 번째 장점은 재사용성이다. 한 번 검증한 함수를 여러 위치에서 호출하면 같은 코드를 다시 작성할 필요가 없다. 예를 들어 평균을 구하는 함수는 1반, 2반, 3반의 점수 배열에 각각 동일하게 사용할 수 있다. 평균 계산 방법을 바꿔야 할 경우에도 함수 내부 한 곳만 수정하면 모든 호출 부분에 동일한 변경이 반영된다. 코드 중복이 줄어들기 때문에 유지보수 시 수정 누락 가능성도 감소한다.

3. 테스트와 오류 추적

세 번째 장점은 테스트와 디버깅이 쉬워진다는 점이다. 프로그램을 함수 단위로 나누면 각 함수에 특정 입력을 주고 예상한 출력이 나오는지 독립적으로 확인할 수 있다. 예를 들어 find_max 함수에 {70, 90, 85}를 넣었을 때 90이 반환되는지만 먼저 검사할 수 있다. 문제가 발견되면 전체 프로그램을 한 줄씩 추적하기보다 해당 기능의 함수에 집중하면 된다. 여러 사람이 함께 개발할 때에도 팀원별로 함수를 나누어 작업한 뒤 정해진 함수 원형에 맞춰 결합할 수 있어 협업에 유리하다.

장점	함수를 쓰지 않을 때	함수를 사용할 때
가독성	main에 여러 기능이 섞여 흐름 파악이 어려움	기능별 이름으로 전체 구조를 쉽게 파악
재사용	같은 코드를 복사하여 반복 작성	한 번 만든 함수를 여러 번 호출
수정	복사된 모든 위치를 각각 수정	함수 한 곳 수정으로 일관되게 반영
테스트	전체 실행 결과에서 원인 추적	함수 단위로 입력·출력 검사 가능
협업	코드 영역이 뒤섞이기 쉬움	함수 단위로 역할을 분담하기 쉬움

IV. 예시: 학생 점수 처리 프로그램

1. 함수 미사용 방식

학생 5명의 점수에서 합계와 최고점을 구한다고 가정해 보자. 함수를 사용하지 않으면 main 안에 입력, 합계 계산, 최고점 탐색, 평균 계산, 출력이 모두 들어간다. 다른 반의 점수를 처리할 때 비슷한 반복문을 다시 작성하게 되고, 계산 방법이 바뀌면 복사된 코드들을 각각 수정해야 한다.

2. 함수 사용 방식과 비교

합계와 최고점 계산을 함수로 분리하면 main은 작업 순서만 보여주고 세부 계산은 각 함수가 담당한다. 배열과 원소 수를 매개변수로 받도록 만들면 다른 학급이나 다른 크기의 데이터에도 같은 함수를 재사용할 수 있다.

[함수로 분리한 핵심 예시]

```
int sum_scores(const int a[], int n) {
    int sum = 0;
    for (int i = 0; i < n; i++) sum += a[i];
    return sum;
}


int main(void) {
    int scores[5] = {78, 92, 84, 69, 88};
    int total = sum_scores(scores, 5);
    printf("avg=%.1f\n", total / 5.0);
}
```

함수화의 핵심은 코드 줄 수를 줄이는 데 있지 않다. 같은 계산을 반복해서 쓰거나 요구사항이 바뀔 때, 검증된 함수를 재사용하고 한 곳만 수정할 수 있어 코드 중복과 수정 누락을 줄이는 데 의미가 있다

V. 결론

함수 없이도 C 프로그램을 작성할 수 있지만, 프로그램 규모가 커질수록 코드 중복과 복잡성이 빠르게 증가한다. 함수를 사용하면 기능별 모듈화가 가능해져 main의 흐름이 명확해지고, 동일 기능을 여러 곳에서 재사용할 수 있으며, 수정과 테스트도 함수 단위로 수행할 수 있다. 학생 점수 처리 예시에서도 합계와 최고점 기능을 분리함으로써 다른 데이터에 동일 코드를 적용하기 쉬워졌다. 따라서 함수는 단순한 문법 요소가 아니라 프로그램을 체계적으로 설계하고 유지보수하기 위한 핵심 도구라고 할 수 있다.

VI. 참고문헌

Kernighan, B. W. & Ritchie, D. M. (1988). The C Programming Language (2nd ed.). Prentice Hall.

King, K. N. (2008). C Programming: A Modern Approach (2nd ed.). W. W. Norton & Company.

Deitel, P. & Deitel, H. (2015). C How to Program (8th ed.). Pearson.

⚠ 주의사항

□ 과제 수정이 어려우신가요?

국민원격교육관리센터에서 1:1 과제 수정 도움을 드립니다. 아래 연락처로 문의해 주세요.

□ 전화: 050-7878-7044

□ 카카오톡: @국민원격교육관리센터

□ 운영시간: 평일 10:00~19:00

□ 웹사이트: korealms.co.kr