Workbook: C 활용 문법 Review

목표: C 활용 문법을 확인하고 스스로 복습하며 이해도 점검

구성:

1. C 언어 기본
2. 연산자(Operators)
3. 제어문(Control Statements)
4. 함수(Function)
5. 포인터(Pointer)
6. 동적 메모리 할당(Dynamic Memory Allocation)
7. 파일 입출력(File I/O)
8. 구조체(Struct)

1. C 언어 기본

C 언어란?

- ✓ 1972년 데니스 리치(Dennis Ritchie) 가 개발한 프로그래밍 언어
- ✓ 운영체제(예: Unix, Linux, Windows 커널), 시스템 프로그래밍, 임베디드 시스템(예: 마이크로컨트롤러, IoT 기기), 게임 개발(예: 그래픽 엔진), 데이터베이스 시스템(예: MySQL) 등에 널리 사용
- ✓ 절차적 프로그래밍 언어(Procedural Language)로 함수와 제어문 중심 구조
- ✓ 빠르고 효율적인 성능을 제공하며, 저수준 메모리 제어 기능 제공

C 언어의 주요 특징

- ✓ 이식성(Portability): 운영 체제에 관계 없이 실행 가능
- ✓ 저수준 제어: 메모리 관리 (포인터, 동적 할당) 가능
- ✓ 빠른 실행 속도: 하드웨어와 직접 상호작용
- ✓ 모듈화: 함수와 라이브러리를 활용하여 코드 관리

● 개념 설명

C 프로그램은 일반적으로 다음과 같은 기본적인 구조를 가진다.

```
#include <stdio.h>  // 헤더 파일 포함
#define TEXT "c언어"  // 매크로 정의
int main() {  // 메인 함수 시작
    printf(TEXT);  // 출력
    int num;
    printf("\nEnter number: ");  // 사용자 입력 안내
    scanf("%d", &num);  // 입력
    printf("You entered: %d\n", num);  // 입력된 값 출력
    return 0;  // 프로그램 종료
}
```

● 코드 분석

구성 요소	설명
#include <stdio.h>	**표준 입출력 라이브러리**포함 (printf, scanf 사용)
#define TEXT "c언어"	**매크로 정의**(TEXT는 "c언어"로 대체됨)
int main()	프로그램의 시작점 (메인 함수)
printf(TEXT);	TEXT를 출력 ("c언어"가 출력됨)
scanf("%d", &num);	정수를 입력받아 num에 저장
return 0;	프로그램 종료

● 확인 문제

[Q1] 다음 코드에서 오류가 발생하는 부분을 찾아 수정하세요.

```
#include <stdio.h>
#define GREETING "Hello, World!"
int main() {
    printf(GREETING);
    int number;
    scanf("%d", number);
    return 0;
}
```

[Q2] 다음 코드를 실행하면 어떤 결과가 출력되는지 설명하시오.

```
#include <stdio.h>
#define X 5
#define Y X + 2
int main() {
    printf("%d", Y * 2);
}
```

2. 연산자 (Operators)

C 언어에서 연산자(Operator) 는 값을 조작하는 데 사용되는 기호에 해당
연산자는 크게 산술, 관계, 논리, 비트, 대입, 증감, 조건, sizeof 등으로 구분

산술 연산자 (Arithmetic Operators)

● 개념 설명

기본적인 사칙연산을 수행하는 연산자

연산자	설명	예제 (a = 10, b = 3)	결과
+	덧셈	a + b	13
-	뺄셈	a - b	7
*	곱셈	a * b	30
/	나눗셈	a / b	3 (정수 나눗셈)
%	나머지	a % b	1

● 예제 코드

```
#include <stdio.h>
int main() {
    int a = 10, b = 3;
    printf("a + b = %d\n", a + b);
    printf("a - b = %d\n", a - b);
    printf("a * b = %d\n", a * b);
    printf("a / b = %d\n", a / b);  // 정수 나눗셈이므로 몫만 출력됨
    printf("a %% b = %d\n", a % b);  // 나머지 연산자 (%%는 printf에서 % 출력)
    return 0;
}
```

● 확인 문제

[Q3] 다음 코드를 실행하면 어떤 결과가 출력되는지 설명하시오.

```
int x = 17, y = 5;
printf("%d %d %d", x / y, x % y, x * y);
```

관계 연산자 (Relational Operators)

● 개념 설명

두 값을 비교하여 참(1) 또는 거짓(0) 반환

연산자	설명	예제 (a = 5, b = 10)	결과
==	같음	a == b	0(거짓)
!=	다름	a != b	1(참)
>	초과	a > b	0(거짓)
<	미만	a < b	1(참)
>=	이상	a >= b	0(거짓)
<=	이하	a <= b	1(참)

● 예제 코드

```
#include <stdio.h>
int main() {
    int a = 5, b = 10;
    printf("a == b: %d\n", a == b);
    printf("a != b: %d\n", a != b);
    printf("a > b: %d\n", a > b);
    printf("a < b: %d\n", a < b);
    return 0;
}
```

● 확인 문제

[Q4] 다음 코드를 실행하면 어떤 결과가 출력되는지 설명하시오.

```
int a = 7, b = 7;
printf("%d %d", a >= b, a != b);
```

논리 연산자 (Logical Operators)

● 개념 설명

논리적인 참(1) 또는 거짓(0)을 반환하는 연산자

연산자	설명	예제 (a = 1, b = 0)	결과
&&	AND (둘 다 참이면 참)	a && b	0
||	OR (둘 중 하나라도 참이면 참)	a || b	1
!	NOT (참 ↔ 거짓 반전)	!a	0

● 예제 코드

```
#include <stdio.h>
int main() {
    int a = 1, b = 0;
    printf("a && b: %d\n", a && b);
    printf("a || b: %d\n", a || b);
    printf("!a: %d\n", !a);
    return 0;
}
```

● 확인 문제

[Q5] 다음 코드를 실행하면 어떤 결과가 출력되는지 설명하시오.

```
int x = 5, y = 0;
printf("%d %d", (x > 0) && (y == 0), (x < 0) || (y > 0));
```

증감 연산자 (Increment & Decrement)

● 개념 설명

변수 값을 1 증가 또는 감소시키는 연산자

연산자	설명	예제 (a = 5)	결과
++a	전위 증가(먼저 증가)	int x = ++a;	x = 6, a = 6
a++	후위 증가(사용 후 증가)	int x = a++;	x = 5, a = 6
--a	전위 감소	int x = --a;	x = 4, a = 4
a--	후위 감소	int x = a--;	x = 5, a = 4

● 예제 코드

```
#include <stdio.h>
int main() {
    int a = 5, b = 5;
    printf("Before: a = %d, b = %d\n", a, b);
    int x = ++a;  // 전위 증가: a를 먼저 증가시키고 x에 대입
    int y = b++;  // 후위 증가: y에 b를 먼저 대입하고 b를 증가
    printf("After: a = %d, x = %d\n", a, x);  // a=6, x=6
    printf("After: b = %d, y = %d\n", b, y);  // b=6, y=5
    return 0;
}
```

● 확인 문제

[Q6] 다음 코드를 실행하면 어떤 결과가 출력되는지 설명하시오.

```
int a = 3;
printf("%d %d %d", ++a, a++, a);
```

3. 제어문(Control Statements)

C 언어에서 프로그램의 흐름을 제어하는 문법을 제어문이라 한다.

- 조건문: if-else, switch-case
- 반복문: for, while, do-while

if-else 문

● 개념 설명

조건에 따라 프로그램의 흐름을 제어하는 제어문

● 예제 코드

```
int num = 10;
if (num > 0) {
    printf("Positive\n");
} else {
    printf("Negative\n");
}
```

● 확인 문제

[Q7] 다음 코드를 실행하면 어떤 결과가 출력되는지 설명하시오.

```
int x = 5;
if (x > 2) {
    printf("A");
}
if (x < 10) {
    printf("B");
} else {
    printf("C");
}
```

switch-case 문

● 개념 설명

하나의 변수를 여러 개의 경우(case)와 비교하여 실행할 코드를 결정하는 제어문

- ✓ case 문 뒤에는 break;를 사용하여 실행을 중단해야 함
- ✓ default 문을 추가하면 모든 case에 해당하지 않는 경우 실행됨
- ✓ break;를 사용하지 않으면 다음 case 문까지 실행됨
- ✓ 여러 case를 묶어 사용할 수도 있음

● 예제 코드

```c
#include <stdio.h>
int main() {
    int day;
    printf("Enter a day number (1-7): ");
    scanf("%d", &day);
    switch (day) {
        case 1:
            printf("Monday\n");
            break;
        case 2:
            printf("Tuesday\n");
            break;
        case 3:
            printf("Wednesday\n");
            break;
        case 4:
            printf("Thursday\n");
            break;
        case 5:
            printf("Friday\n");
            break;
        case 6:
            printf("Saturday\n");
            break;
        case 7:
            printf("Sunday\n");
            break;
        default:
            printf("Invalid input! Enter a number between 1 and 7.\n");
    }
    return 0;
}
```

```c
#include <stdio.h>
int main() {
    int grade;
    printf("Enter your grade (1-5): ");
    scanf("%d", &grade);
    switch (grade) {
        case 1:
        case 2:
            printf("Needs Improvement\n");
            break;
        case 3:
        case 4:
            printf("Good Job\n");
            break;
        case 5:
            printf("Excellent!\n");
            break;
        default:
            printf("Invalid Grade\n");
    }
    return 0;
}
```

● 확인 문제

[Q8] 다음 코드를 실행하면 어떤 결과가 출력되는지 설명하시오.

```
#include <stdio.h>
int main() {
    int num = 2;
    switch (num) {
        case 1:
            printf("One\n");
        case 2:
            printf("Two\n");
        case 3:
            printf("Three\n");
            break;
        default:
            printf("Default\n");
    }
    return 0;
}
```

for 문

● 개념 설명

특정 코드 블록을 여러 번 실행할 때 사용

반복 횟수가 정해진 경우에 사용

- ✓ 초기식(Initialization): 반복을 시작하기 전에 실행 (반복 변수 선언 및 초기화)
- ✓ 조건식(Condition): 반복을 계속할지 결정 (조건이 참이면 실행, 거짓이면 종료)
- ✓ 증감식(Update): 반복 변수를 변경하여 조건을 만족시키는지 확인

작동 순서

1. 초기식이 처음 한 번 실행됨
2. 조건식이 참이면 반복 실행 → 거짓이면 반복 종료
3. 코드 블록 실행
4. 증감식 실행 후 조건식 다시 확인
5. 조건이 참이면 반복 계속, 거짓이면 종료

● 예제 코드

```
#include <stdio.h>
int main() {
    for (int i = 1; i <= 10; i++) {
        printf("%d ", i);
    }
    return 0;
}
```

● 확인 문제

[Q9] 다음 코드를 실행하면 어떤 결과가 출력되는지 설명하시오.

```
#include <stdio.h>
int main() {
    for (int i = 10; i >= 1; i -= 2) {
        printf("%d ", i);
    }
    return 0;
}
```

[Q10] 1~100까지 5의 배수만 출력하는 for 문을 작성하세요.

```
#include <stdio.h>
int main() {
    for (__________) {
        printf("%d ", i);
    }
    return 0;
}
```

while 문

● 개념 설명

조건이 참(true)인 동안 코드 블록을 계속 실행하는 반복문

- ✓ 반복 횟수가 정해지지 않은 경우에 사용하기 적합
- ✓ for 문과 달리 초기화, 조건식, 증감식을 따로 설정해야 함

중요 주의 사항

- ✓ while 문을 사용할 때는 반복이 종료될 조건을 명확히 설정
- ✓ 무한 루프(Infinite Loop) 발생에 유의

● 예제 코드

```
#include <stdio.h>
int main() {
    int i = 2;
    while (i <= 20) {
        printf("%d ", i);
        i += 2;  // 짝수로 증가
    }
    return 0;
}
```

● 확인 문제

[Q11] 다음 코드를 실행하면 어떤 결과가 출력되는지 설명하시오.

```
#include <stdio.h>
int main() {
    int i = 10;
    while (i >= 1) {
        printf("%d ", i);
        i -= 2;
    }
    return 0;
}
```

[Q12] 무한 루프를 수정하여 1~20까지 출력 후 프로그램이 정상적으로 종료되도록 만드세요.

```
#include <stdio.h>
int main() {
    int i = 1;
    while (20) {
        printf("%d ", i);
    }
    return 0;
}
```

do-while 문

● 개념 설명

반드시 한 번은 실행되는 반복문

while 문과 다르게, 조건을 나중에 검사하므로 최소 한 번은 실행

기본 구조:

```
do {
    // 반복 실행할 코드
} while (조건식);
```

작동 순서:

1. 코드 블록 실행
2. 조건 검사
3. 조건이 참이면 반복, 거짓이면 종료

반드시 한 번 실행됨 → 조건이 처음부터 거짓이어도 최소 1회 실행

● 예제 코드

```
#include <stdio.h>
int main() {
    int password;
    do {
        printf("Enter password (1234 to unlock): ");
        scanf("%d", &password);
    } while (password != 1234);
    printf("Access granted!\n");
    return 0;
}
```

● 확인 문제

[Q13] 다음 코드를 실행하면 어떤 결과가 출력되는지 설명하시오.

```
#include <stdio.h>
int main() {
    int i = 10;
    do {
        printf("%d ", i);
        i -= 2;
    } while (i >= 1);
    return 0;
}
```

[Q14] 1부터 30까지 3의 배수를 출력하는 do-while 문을 작성하세요.

```
#include <stdio.h>
int main() {

}
```

4. 함수(Function)

● 개념 설명

함수는 코드의 재사용성을 높이고 프로그램을 모듈화하는 중요한 개념

- ✓ 함수 선언: 반환형 함수이름(매개변수);
- ✓ 함수 정의: 함수의 실제 구현
- ✓ 함수 호출: main()에서 실행

함수에 인자를 전달하는 방식은 Call by Value(값 호출) 과 Call by Reference(참조 호출)

- Call by Value (값 호출)
  - ✓ 매개변수의 값을 복사하여 전달
  - ✓ 원본 변수에는 영향을 미치지 않음
- Call by Reference (참조 호출)
  - ✓ 변수의 메모리 주소를 전달하여 원본 값을 직접 변경할 수 있음
  - ✓ 포인터(pointer)를 사용하여 변수의 주소를 전달

차이점	Call by Value (값 호출)	Call by Reference (참조 호출)
전달 방식	변수의 **값을 복사**하여 전달	변수의 **주소를 전달**
원본 데이터 변경 여부	**변경되지 않음**	**변경됨**
사용 방식	기본 변수 타입 사용 (int a)	포인터 사용 (int *a)
메모리 사용량	원본 값의 복사본 생성	원본 주소를 사용하여 메모리 절약
함수 내 수정 효과	원본 값에 영향 없음	원본 값이 직접 변경됨
사용 예	값이 변경되지 않아야 하는 경우 (예: 단순 출력 함수)	값이 변경되어야 하는 경우 (예: 정렬, 배열 수정)

● 예제 코드

```
#include <stdio.h>
int multiply(int a, int b);  // 함수 선언
int main() {
    int result = multiply(4, 5);
    printf("%d\n", result);
    return 0;
}

int multiply(int a, int b) {  // 함수 정의
    return a * b;
}
```

```
#include <stdio.h>    // Call by Reference (참조 호출)
void changeValue(int *a) {  // 포인터를 사용하여 참조
    *a = 10;  // 포인터를 통해 원본 값 변경
}

int main() {
    int num = 5;
    changeValue(&num);          // num의 주소를 전달
    printf("num: %d\n", num);   // 원본 값 변경됨
    return 0;
}
```

● 확인 문제

[Q15] 다음 코드에서 오류가 발생하는 부분을 찾아 수정하세요.

```
#include <stdio.h>
int add(int a, int b);  // 함수 선언
int main() {
    int result = add(4, 5);
    printf("%d\n", result);
    return 0;
}

int add(int a, int b); {
    return a + b;
}
```

[Q16] 다음 코드를 실행하면 어떤 결과가 출력되는지 설명하시오.

```
#include <stdio.h>
void modify(int a) {
    a = 10;
}
void modifyRef(int *b) {
    *b = 20;
}

int main() {
    int x = 5, y = 5;
    modify(x);
    modifyRef(&y);
    printf("x: %d, y: %d\n", x, y);
    return 0;
}
```

5. 포인터(Pointer)

● 개념 설명

포인터는 메모리 주소를 저장하는 변수

* 기호를 사용하여 선언하며, & 기호를 사용하여 변수의 주소를 가져옴

- ✓ int *ptr; → 정수를 가리키는 포인터
- ✓ ptr = # → 변수 num의 주소 저장
- ✓ *ptr → 포인터가 가리키는 값

● 예제 코드

```
int num = 10;        // 일반 변수
int *ptr = &num;     // 포인터 변수, num의 주소 저장
printf("%d", *ptr);  // 10 출력
```

```
#include <stdio.h>
int main() {
    int num = 5;
    int *ptr = &num;  // num의 주소를 ptr에 저장
    printf("Before: num = %d\n", num);    // 5가 출력
    *ptr = 20;  // 포인터를 이용해 num의 값을 변경
    printf("After: num = %d\n", num);     // 20이 출력
    return 0;
}
```

● 확인 문제

[Q17] 다음 코드를 실행하면 어떤 결과가 출력되는지 설명하시오.

```
#include <stdio.h>
int main() {
    int num = 25;
    int *ptr = &num;
    printf("%d %d %p %p\n", num, *ptr, &num, ptr);
    return 0;
}
```

6. 동적 메모리 할당(Dynamic Memory Allocation)

● 개념 설명

프로그램 실행 중에 메모리를 할당하고 해제하는 기능

[참고] 정적(Static) 메모리 할당: 컴파일 시 크기가 고정됨 (예: int arr[10];)

동적(Dynamic) 메모리 할당: 런타임 중 크기 변경 가능

- ✓ malloc(): 메모리 할당
- ✓ calloc(): 초기화된 메모리 할당
- ✓ free(): 메모리 해제

stdlib.h 헤더 파일을 포함해야 사용할 수 있음

함수	설명	반환 값
malloc(size)	지정한 크기(size)만큼 메모리를 할당	성공 시 포인터 반환, 실패 시 NULL
calloc(n, size)	n개의 요소를 초기화 후 할당	성공 시 포인터 반환, 실패 시 NULL
realloc(ptr, new_size)	기존 메모리 크기를 변경	성공 시 새 포인터 반환, 실패 시 NULL
free(ptr)	할당된 메모리를 해제	없음

● 예제 코드

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int *ptr;
    // 정수형 크기만큼 동적 메모리 할당
    ptr = (int*) malloc(sizeof(int));
    if (ptr == NULL) {    // 메모리 할당 실패 시 종료
        printf("Memory allocation failed\n");
        return 1;
    }
    *ptr = 42;  // 할당된 메모리에 값 저장
    printf("Allocated Value: %d\n", *ptr);
    free(ptr);  // 동적 메모리 해제
    return 0;
}
```

● 확인 문제

[Q18] 다음 코드를 실행하면 어떤 결과가 출력되는지 설명하시오.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int *arr;
    int size = 5;
    arr = (int*) malloc(size * sizeof(int));
    if (arr == NULL) {
        printf("Memory allocation failed\n");
        return 1;
    }
    for (int i = 0; i < size; i++) {
        arr[i] = i + 1;
        printf("%d ", arr[i]);
    }
    free(arr);
    return 0;
}
```

7. 파일 입출력(File I/O)

● 개념 설명

데이터를 저장하고 읽는 데 사용

stdio.h 헤더 파일을 포함해야 사용 가능

- ✓ fopen(), fclose(): 파일 열기/닫기
- ✓ fprintf(), fscanf(): 파일에 쓰기/읽기
- ✓ fputs() → 문자열 쓰기 / fgets() → 파일에서 한 줄씩 읽기
- ✓ fwrite() → 바이너리 데이터 쓰기 / fread() → 바이너리 데이터 읽기

파일 모드(mode) 를 통해 파일을 읽기/쓰기/추가

- ✓ "r": 읽기 모드 (파일이 존재해야 함)
- ✓ "w": 쓰기 모드 (파일이 없으면 생성, 있으면 내용 삭제 후 새로 씀)
- ✓ "a": 추가 모드 (파일이 없으면 생성, 있으면 기존 내용 유지 후 추가)
- ✓ "r+": 읽기/쓰기 모드 (기존 파일 유지)
- ✓ "w+": 읽기/쓰기 모드 (파일이 없으면 생성, 있으면 기존 내용 삭제)
- ✓ "a+": 읽기/쓰기 모드 (파일이 없으면 생성, 있으면 기존 내용 유지 후 추가)

● 예제 코드

```
#include <stdio.h>
int main() {
    FILE *fp = fopen("data.txt", "w");    // "data.txt" 파일을 쓰기 모드로 열기
    if (fp == NULL) {
        printf("File open error\n");
    }
    fprintf(fp, "Hello, World!");
    fprintf(fp, "This is a c text file.\n");
    fclose(fp);
    return 0;
}
```

실행 후 data.txt 파일 내용:

```
Hello, World!
This is a c text file.
```

● 확인 문제

[Q19] 다음 코드를 실행하면 어떤 결과가 출력되는지 설명하시오.

```
#include <stdio.h>
int main() {
    FILE *file = fopen("test.txt", "w");
    if (file == NULL) {
        printf("파일을 열 수 없습니다.\n");
        return 1;
    }
    fprintf(file, "C Programming\nFile I/O Test\n");
    fclose(file);
    file = fopen("test.txt", "r");
    char line[50];
    while (fgets(line, sizeof(line), file) != NULL) {
        printf("%s", line);
    }
    fclose(file);
    return 0;
}
```

8. 구조체(Struct)

- 개념 설명

서로 다른 데이터 타입을 하나의 그룹으로 묶을 수 있는 사용자 정의 자료형

- ✓ 여러 개의 변수(필드)를 하나로 묶어 관리
- ✓ struct 키워드를 사용하여 정의
- ✓ 구조체 선언 후 변수를 생성하여 사용 가능
- ✓ . 연산자를 사용하여 필드 접근

- 예제 코드

```
#include <stdio.h>
// 구조체 정의
struct Student {
    char name[10];
    int age;
    float grade;
};
int main() {
    struct Student s1 = {"Kim", 21, 4.3};  // 구조체 변수 초기화
    printf("이름: %s\n", s1.name);
    printf("나이: %d\n", s1.age);
    printf("학점: %.1f\n", s1.grade);
    return 0;
}
```

```
// 구조체와 포인터
#include <stdio.h>
struct Student {
    char name[50];
    int age;
    float grade;
};
int main() {
    struct Student s1 = {"Kim", 21, 4.3};
    struct Student *ptr = &s1;  // 구조체 포인터 선언
    printf("이름: %s\n", ptr->name);
    printf("나이: %d\n", ptr->age);
    printf("학점: %.1f\n", ptr->grade);
    return 0;
}
```

```
// 구조체 배열
#include <stdio.h>
struct Student {
    char name[50];
    int age;
    float grade;
};
int main() {
    struct Student students[3] = {
        {"Kim", 20, 4.0},
        {"Lee", 22, 3.8},
        {"Park", 21, 4.2}
    };
    for (int i = 0; i < 3; i++) {
        printf("이름: %s, 나이: %d, 학점: %.1f\n",
               students[i].name, students[i].age, students[i].grade);
    }
    return 0;
}
```

```c
// 구조체와 동적 메모리 할당 (malloc)
#include <stdio.h>
#include <stdlib.h>
struct Student {
    char name[50];
    int age;
    float grade;
};
int main() {
    struct Student *s = (struct Student*) malloc(sizeof(struct Student));
    if (s == NULL) {
        printf("메모리 할당 실패!\n");
        return 1;
    }

    // 구조체 포인터를 사용하여 값 설정
    printf("이름 입력: ");
    scanf("%s", s->name);
    printf("나이 입력: ");
    scanf("%d", &s->age);
    printf("학점 입력: ");
    scanf("%f", &s->grade);

    printf("\n학생 정보\n이름: %s\n나이: %d\n학점: %.1f\n",
           s->name, s->age, s->grade);

    free(s);  // 동적 메모리 해제
    return 0;
}
```

● 확인 문제

[Q20] 다음 코드에서 -> 연산자를 사용하여 구조체 포인터로 접근하도록 수정하세요.

```c
#include <stdio.h>
struct Car {
    char brand[50];
    int year;
};

int main() {
    struct Car car = {"Hyundai", 2025};
    struct Car *ptr = &car;

    printf("%s %d\n", __________, __________);

    return 0;
}
```

- 수고하셨습니다 -