1. 자료형과 연산 기초

1.1 자료형의 정의와 필요성

자료형은 프로그래밍에서 데이터가 메모리에 저장되는 방법과 크기를 결정하는 중요한 요소이다. C 언어에서 자료형은 데이터의 크기와 유형을 명확히 규정하여 프로그램이 정확하게 데이터 처리를 수행할 수 있도록 돕는다. 자료형이 명시되지 않으면 메모리에 저장된 데이터가 어떻게 해석되어야 할지 불분명해져 오류가 발생할 수 있다.

기본 자료형 개요

1. 정수형 (`int`)

- 정수를 표현하며, 보통 4바이트 크기를 가지므로 -2,147,483,648에서 2,147,483,647까지의 범위를 저장할 수 있다.

2. 부동 소수점형 (`float`)

- 소수점 이하의 실수 값을 표현하며, 4바이트 크기이고, 약 7자리까지의 유효 자릿수를 가지는 실수를 표현할 수 있다.

3. 문자형 (`char`)

- 한 글자를 저장하는 데 사용되며, 1바이트 크기로 각 문자는 ASCII 코드로 변환되어 저장된다.

자료형을 정확히 이해하면 효율적으로 메모리를 관리하고 프로그램의 성능을 최적화할 수 있다.

1.2 변수와 상수

변수는 데이터를 저장하기 위한 메모리 공간의 이름으로, 코드에서 참조할 때 사용된다. 상수는 초기화 이후 값을 변경할 수 없으며, 변경하지 않을 데이터에는 상수를 사용하여 코드의 가독성을 높일 수 있다.

1. 변수 선언과 초기화

변수는 사용 전에 선언되어야 하며, 선언 시 초기값을 지정할 수 있다.



2. 상수 선언

상수는 `const` 키워드를 사용하여 선언하며, 프로그램 내에서 변경되지 않는 데이터를 정의할 때 유용하다.



상수는 코드의 의미를 명확히 전달하고, 의도치 않은 수정이 발생하지 않도록 보호하는 데 유용하다.

1.3 표준 입출력

C 언어에서는 `printf`와 `scanf` 함수를 통해 데이터 출력과 입력을 처리한다. 이들 함수는 `<stdio.h>` 헤더 파일에 정의되어 있어, 입출력을 사용하려면 반드시 해당 헤더 파일을 포함해야 한다.

1. `printf` 함수

- `printf`는 화면에 데이터를 출력하는 함수로, 서식 문자열을 사용하여 여러 데이터 타입을 형식에 맞게 출력할 수 있다.



2. `scanf` 함수

- `scanf`는 사용자로부터 입력을 받는 함수로, 특정 변수를 참조하여 데이터를 입력받는다.

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

3. 서식 문자열

- `printf`와 `scanf`는 각 자료형에 맞는 서식 지정자를 사용한다.

- `%d`: 정수형

- `%f`: 실수형

- `%c`: 문자형

- `%s`: 문자열형

1.4 기초 연산자와 식

연산자는 데이터 간의 연산을 수행하는 데 사용되며, C 언어에서는 다양한 연산자를 제공한다. C 언어의 연산자는 크게 산술, 관계, 논리 연산자로 나뉘며, 각 연산자는 고유한 우선순위를 가진다.

1. 산술 연산자

- 기본적인 수학 연산을 수행하며, `+`, `-`, `\*`, `/`, `%`가 대표적이다.

- `%` 연산자는 나머지 값을 반환하며, 주로 정수형 값에 대해 사용된다.

텍스트, 폰트, 스크린샷, 화이트이(가) 표시된 사진

자동 생성된 설명

2. 관계 연산자

- 두 값을 비교하는 연산자로, 비교의 결과를 1(참) 또는 0(거짓)으로 반환한다. 관계 연산자는 조건문 등에서 사용된다.

- 예시: `==`, `!=`, `<`, `>`, `<=`, `>=`

텍스트, 폰트, 라인, 친필이(가) 표시된 사진

자동 생성된 설명

3. 논리 연산자

- 여러 조건을 결합하여 하나의 논리적 판단을 수행하는 연산자로, 대표적인 논리 연산자는 `&&`(AND), `||`(OR), `!`(NOT)가 있다.

텍스트, 폰트, 친필, 라인이(가) 표시된 사진

자동 생성된 설명

4. 연산자 우선순위

- 여러 연산자를 사용할 때에는 우선순위에 따라 연산이 수행된다. 산술 연산자가 관계 연산자보다 우선순위가 높고, 관계 연산자가 논리 연산자보다 우선순위가 높다.



1.5 형변환 (Type Conversion)

형변환은 서로 다른 자료형을 변환하는 과정으로, C 언어에서는 자동 형변환과 명시적 형변환 두 가지가 있다.

1. 자동 형변환 (Implicit Type Conversion)

- 더 높은 정밀도의 자료형으로 자동 변환이 이루어진다. 예를 들어, `int`와 `float`이 함께 사용되면 `int`가 자동으로 `float`로 변환된다.

2. 명시적 형변환 (Explicit Type Conversion)

- 개발자가 형변환을 강제로 지정하는 것으로, `(자료형)`을 사용해 명시한다.

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

3. 형변환의 중요성

- 형변환은 자료 손실을 방지하고, 의도한 연산 결과를 얻기 위해 중요하다. 특히 정수와 실수의 연산에서 자주 활용된다.

1.6 오류의 분류

C 언어 프로그램에서 발생할 수 있는 주요 오류 유형은 다음과 같다:

1. 구문 오류 (Syntax Error)

- 문법을 잘못 사용하여 발생하며, 주로 컴파일 단계에서 감지된다.

- 예: 세미콜론(`;`) 누락, 잘못된 키워드 사용 등.

2. 논리 오류 (Logical Error)

- 프로그램이 의도와 다르게 동작하게 만드는 오류로, 컴파일 시점에는 발견되지 않으며 실행 후 결과가 예상과 다른 경우 발생한다.

- 예: 잘못된 수식이나 조건문 사용으로 인한 오류.

3. 실행 오류 (Run-Time Error)

- 프로그램 실행 도중에 발생하는 오류로, 주로 잘못된 입력이나 메모리 접근 오류가 원인이다.

- 예: 배열 인덱스 초과, 0으로 나누기 등.

자료형과 연산 기초는 C 언어의 기초로, 메모리 관리를 효율화하고 프로그램 오류를 줄이는 데 중요한 역할을 한다. 이러한 기초 개념을 이해하는 것은 C 언어의 고급 주제로 나아가기 위한 필수적인 첫걸음이다.

2. 제어 구조 기초

2.1 제어 구조와 조건문

C 언어에서 제어 구조는 프로그램의 흐름을 제어하여 논리적이고 유연한 프로그램을 작성하는 데 필수적이다. 프로그램의 실행 순서는 기본적으로 첫 번째 문장부터 순차적으로 진행되지만, 제어 구조를 사용하면 특정 조건에 따라 다른 문장이 실행되도록 조정할 수 있다. 제어 구조의 핵심에는 조건문과 반복문이 있으며, 이를 통해 프로그램의 논리 흐름을 효과적으로 관리할 수 있다.

2.2 조건문 (Conditional Statements)

조건문은 특정 조건이 참인지 거짓인지에 따라 프로그램의 실행 흐름을 분기하는 데 사용된다. C 언어의 조건문에는 if, if-else, else if, 그리고 switch 문이 있다.

1. if 문

- if 문은 주어진 조건식이 참일 경우에만 블록 내의 문장을 실행한다. 조건이 거짓이면 다음 문장으로 넘어간다.

텍스트, 폰트, 라인, 친필이(가) 표시된 사진

자동 생성된 설명

2. if-else 문

- if-else 문은 주어진 조건이 참이면 if 블록을, 거짓이면 else 블록을 실행한다. 이를 통해 조건에 따라 두 가지 경로로 분기할 수 있다.

텍스트, 폰트, 친필, 라인이(가) 표시된 사진

자동 생성된 설명

3. else if 문

- else if 문은 여러 조건을 나열하여 하나의 조건만 만족할 경우 특정 블록을 실행하도록 한다. 다중 조건이 필요한 경우 유용하다.

텍스트, 폰트, 스크린샷, 친필이(가) 표시된 사진

자동 생성된 설명

4. switch 문

- switch 문은 특정 변수의 값을 기준으로 여러 가지 실행 경로 중 하나를 선택하는 다중 선택 구조이다. 주로 정수형 또는 문자형 값을 조건으로 사용하며, 각 조건은 case 문으로 구분된다.

- break 문을 사용해 각 case 블록이 끝나면 switch 문을 종료하도록 한다. default 블록은 모든 case 조건이 일치하지 않을 때 실행된다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

2.3 반복문 (Loop Statements)

반복문은 특정 조건이 만족되는 동안 동일한 블록을 반복해서 실행하도록 한다. 대표적인 반복문으로는 while, for, do-while 문이 있다.

1. while 문

- while 문은 조건식이 참일 동안 반복해서 블록을 실행한다. 조건이 거짓이면 반복을 종료한다.

폰트, 텍스트, 라인, 친필이(가) 표시된 사진

자동 생성된 설명

2. for 문

- for 문은 주로 반복 횟수가 명확한 경우 사용되며, 초기화, 조건 검사, 증감식을 한 줄에 명시하여 코드가 간결하다.

폰트, 텍스트, 친필, 라인이(가) 표시된 사진

자동 생성된 설명

3. do-while 문

- do-while 문은 while 문과 유사하나, 블록을 한 번 실행한 후 조건을 검사한다. 따라서 조건이 거짓이더라도 최소 한 번은 실행된다.

텍스트, 폰트, 라인, 화이트이(가) 표시된 사진

자동 생성된 설명

4. 무한 반복

- 조건을 항상 참으로 설정하여 반복이 종료되지 않도록 할 수 있으며, 주로 while (1)이나 for (;;)와 같은 구문을 사용한다. 이 경우 반드시 break 문을 통해 특정 조건에서 반복을 종료하도록 해야 한다.

텍스트, 폰트, 친필, 라인이(가) 표시된 사진

자동 생성된 설명

2.4 논리 연산자와 비교 연산자

조건문과 반복문에서 조건식을 작성할 때, 논리 연산자와 비교 연산자를 사용하여 복잡한 조건을 처리할 수 있다.

1. 비교 연산자

- 두 값을 비교하여 참(1) 또는 거짓(0)을 반환한다. 주로 if 문과 함께 사용된다.

- 예시: ==(같음), !=(다름), <, >, <=, >=

텍스트, 폰트, 친필, 화이트이(가) 표시된 사진

자동 생성된 설명

2. 논리 연산자

- 여러 조건식을 결합하여 하나의 결과를 도출할 때 사용한다.

- &&: 두 조건이 모두 참일 때만 참을 반환한다.

- ||: 하나의 조건이라도 참이면 참을 반환한다.

- !: 조건의 참/거짓을 반전시킨다.

텍스트, 폰트, 친필, 라인이(가) 표시된 사진

자동 생성된 설명

2.5 연산자 우선순위와 결합 순서

여러 연산자가 하나의 조건식에 사용될 경우, 연산자의 우선순위와 결합 순서를 이해하는 것이 중요하다. 우선순위가 높은 연산자가 먼저 계산되며, 우선순위가 동일할 경우 결합 순서에 따라 계산이 진행된다.

- 산술 연산자가 가장 높은 우선순위를 가지며, 그 다음으로 관계 연산자, 논리 연산자 순으로 우선순위가 낮아진다.

- 예시:



복잡한 조건식을 처리할 때는 괄호를 사용하여 우선순위를 명확히 지정하는 것이 좋다. 이를 통해 가독성을 높이고 코드가 의도대로 실행되도록 할 수 있다.



2.6 중첩 제어문

조건문과 반복문은 중첩해서 사용할 수 있다. 중첩을 통해 복잡한 조건을 처리하거나, 반복문 안에 조건문을 포함하여 특정 조건에서만 반복을 수행하도록 할 수 있다.

중첩 조건문 예시:

텍스트, 폰트, 스크린샷, 친필이(가) 표시된 사진

자동 생성된 설명

중첩 반복문 예시:

텍스트, 친필, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

중첩 제어문을 활용하면 더 복잡한 논리를 효과적으로 구현할 수 있지만, 너무 깊은 중첩은 코드의 가독성을 저하시킬 수 있으므로 주의해야 한다.

제어 구조는 프로그램의 논리를 구성하는 데 필수적인 요소이며, 이를 통해 유연한 프로그램 설계가 가능하다. 조건문과 반복문의 적절한 활용은 코드의 효율성을 높이고, 유지 보수성을 강화하는 데 중요한 역할을 한다.

3. 함수 기초

3.1 함수의 개념과 필요성

함수는 프로그램을 구성하는 기본 단위로, 코드의 중복을 줄이고 가독성을 높이며 유지보수를 쉽게 하기 위해 사용된다. C 언어에서 함수는 하나의 작업을 수행하는 명령어 집합으로, 주어진 입력을 받아 결과를 반환하거나, 부수적인 효과(예: 화면에 출력)를 통해 프로그램을 진행시킨다. 함수는 주로 코드의 모듈화와 재사용성을 높이는 역할을 한다.

- 장점:

- 코드의 재사용성이 높아지고, 유지 보수가 용이하다.

- 프로그램의 가독성이 증가하며, 디버깅과 테스트가 쉬워진다.

3.2 함수의 정의와 구조

함수는 헤더(Header)와 몸체(Body)로 구성된다. C 언어에서 함수 정의 방식은 반환형 함수명(매개변수 리스트) 형태의 헤더와 중괄호 `{ }`로 감싸인 함수 몸체로 이루어진다.

예시:

폰트, 텍스트, 라인, 화이트이(가) 표시된 사진

자동 생성된 설명

1. 함수 헤더:

- 반환형 (Return Type): 함수가 수행 후 반환하는 값의 데이터형을 나타낸다. 반환값이 없을 경우 `void`를 사용한다.

- 함수 이름 (Function Name): 함수의 식별자로, 함수 호출 시 사용된다.

- 매개변수 리스트 (Parameters): 함수가 입력으로 받을 변수들로, 함수 외부에서 전달된 값을 함수 내부로 전달하는 역할을 한다.

2. 함수 몸체:

- `{ }` 사이에 함수의 실제 코드가 작성되며, 필요한 계산이나 작업을 수행하고 결과를 반환하는 역할을 한다.

- `return` 문을 사용해 함수의 결과를 반환하며, 반환값의 자료형은 함수의 반환형과 일치해야 한다.

3.3 함수 호출 방식

함수를 사용하려면 호출이 필요하다. C 언어의 함수 호출 방식은 Call by Value로, 호출 시 인수의 값을 복사하여 매개변수에 전달한다. 이로 인해 함수 내에서 매개변수 값을 변경해도 원본 변수에는 영향이 없다.

예시:

텍스트, 폰트, 라인, 화이트이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

- Call by Value:

- 함수 호출 시 실제 값이 복사되어 함수 내부의 매개변수에 전달된다.

- 함수 내에서 매개변수 값을 변경해도 원본 값에는 영향을 미치지 않으므로 데이터 보호 측면에서 안전하다.

3.4 반환형과 `return` 문

함수는 수행 후 결과값을 반환하거나, 결과값 없이 종료할 수 있다. `return` 문은 함수의 실행을 종료하고 호출 지점으로 돌아가게 하며, 필요에 따라 값을 반환할 수도 있다.

1. 값 반환하는 함수:

- 반환형이 `int`, `float` 등 특정 자료형일 경우 `return` 문을 통해 값을 반환해야 한다.

- 반환되는 값의 자료형은 함수의 반환형과 일치해야 한다.

텍스트, 폰트, 라인, 스크린샷이(가) 표시된 사진

자동 생성된 설명

2. 값을 반환하지 않는 함수:

- 반환형이 `void`인 함수는 `return` 문을 사용하지 않거나, `return;`으로 함수 종료를 표시할 수 있다.

텍스트, 폰트, 라인, 화이트이(가) 표시된 사진

자동 생성된 설명

3.5 매개변수와 지역 변수

매개변수는 함수 호출 시 외부로부터 전달된 값을 받기 위한 변수이다. C 언어에서 함수 내부에 선언된 변수는 지역 변수로, 함수가 실행되는 동안만 메모리에 존재하며, 함수가 종료되면 자동으로 사라진다.

1. 매개변수:

- 함수 호출 시 인수로 전달되는 값은 함수의 매개변수에 복사된다. 매개변수는 함수 내부에서만 사용할 수 있으며, 함수 외부의 변수에는 영향을 주지 않는다.

폰트, 텍스트, 라인, 스크린샷이(가) 표시된 사진

자동 생성된 설명

2. 지역 변수:

- 함수 내에서 선언된 변수는 해당 함수 내에서만 유효하며, 다른 함수에서는 접근할 수 없다.

- 같은 이름의 변수가 함수 외부에 있어도 지역 변수는 그 외부 변수와는 독립적으로 작동한다.

3.6 전역 변수와 지역 변수

변수는 선언 위치에 따라 전역 변수와 지역 변수로 구분된다.

1. 전역 변수:

- 함수 외부에 선언된 변수로, 프로그램 내 모든 함수에서 접근 가능하다.

- 프로그램이 실행되는 동안 메모리에 존재하며, 여러 함수에서 사용할 수 있다.



2. 지역 변수:

- 함수 내부에서 선언된 변수로, 해당 함수 내에서만 접근할 수 있으며, 함수가 종료되면 메모리에서 사라진다.

텍스트, 폰트, 라인, 화이트이(가) 표시된 사진

자동 생성된 설명

> 전역 변수는 데이터 공유가 용이하지만, 함수 간의 데이터 의존성을 증가시킬 수 있어 사용 시 주의가 필요하다.

3.7 함수의 프로토타입 (Function Prototype)

C 언어에서는 함수 정의 전에 함수의 선언부를 미리 작성할 수 있다. 이를 함수의 프로토타입이라고 하며, 컴파일러가 함수의 반환형과 매개변수를 미리 인식하도록 돕는다.

- 예시:

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

프로토타입을 사용하면 함수 정의 위치와 상관없이 프로그램이 함수를 올바르게 인식할 수 있다.

3.8 재귀 함수 (Recursive Function)

재귀 함수는 자기 자신을 호출하는 함수로, 특정 조건에 따라 스스로 반복해서 호출되며, 종료 조건이 있어야 한다. 재귀 함수는 특정 문제를 반복적인 방식으로 해결할 수 있어, 팩토리얼 계산, 피보나치 수열 등에서 자주 사용된다.

예시: 팩토리얼 계산

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

- 장점: 코드가 간결하고, 복잡한 문제를 논리적으로 표현할 수 있다.

- 단점: 호출이 많아지면 스택 메모리가 초과될 수 있어 성능에 영향을 줄 수 있다.

3.9 함수의 유형

1. 반환값이 있는 함수:

- 특정 값을 반환하며, 반환형을 명시하고 `return` 문을 통해 결과를 호출 지점에 전달한다.

2. 반환값이 없는 함수 (void 함수):

- 반환형을 `void`로 명시하고, 함수 수행 후 특정 결과를 반환하지 않으며 부수적 효과(Side Effect)로 프로그램을 진행시킨다.

3. 매개변수가 없는 함수:

- 매개변수 없이 호출되는 함수로, 특정 작업을 수행하거나 전역 변수를 참조해 작업을 수행한다.

예시:

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

함수는 프로그램을 모듈화하여 코드의 재사용성을 높이고, 복잡한 프로그램을 이해하기 쉽게 만들어 준다. 함수 개념을 잘 이해하고 응용하면 코드의 가독성을 높이고 유지 보수를 용이하게 할 수 있다.

4. 다양한 자료형과 연산

4.1 다양한 자료형과 자료 표현

C 언어에서는 데이터의 표현과 저장을 위해 다양한 자료형을 제공한다. 자료형을 선택할 때는 메모리 사용 효율성과 데이터의 범위를 고려하는 것이 중요하다.

1. 정수형 (Integer Types)

- 정수를 저장하는 자료형으로, 크기와 표현 가능한 값의 범위에 따라 `char`, `short`, `int`, `long`, `long long` 등의 자료형이 있다.

- `unsigned` 정수형은 양수만 표현하여 보다 큰 양의 정수 값을 저장할 수 있다.

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

2. 부동 소수점형 (Floating-Point Types)

- 소수점 이하 값을 포함하는 실수를 표현하며, `float`, `double`, `long double`이 있다. 각각의 자료형은 표현할 수 있는 정밀도와 범위가 다르다.

- `float`는 4바이트로 약 7자리, `double`은 8바이트로 약 15자리의 유효 자릿수를 가진다.



3. 문자형 (Character Type)

- `char`는 1바이트로 단일 문자를 저장할 수 있으며, ASCII 코드를 사용하여 문자와 숫자를 표현한다.



4. 배열 (Array)와 문자열 (String)

- 배열은 동일한 자료형의 데이터를 연속된 메모리 공간에 저장하는 자료구조이며, 문자 배열은 문자열을 표현하는 데 사용된다.

- C 언어에서 문자열은 `char` 배열로 표현되며, 마지막에 `\0` (null 문자)로 끝난다.



4.2 자료형과 메모리 크기

자료형은 고유한 메모리 크기를 가지며, 각 자료형에 할당된 메모리 크기는 저장할 수 있는 값의 범위를 결정한다.

- C 언어에서 `sizeof` 연산자를 사용해 자료형 또는 변수의 메모리 크기를 확인할 수 있다.

- 주요 자료형의 메모리 크기 예시:

- `char` : 1바이트

- `short` : 2바이트

- `int` : 4바이트

- `long` : 4 또는 8바이트 (시스템에 따라 다름)

- `float` : 4바이트

- `double` : 8바이트

- `long double` : 10 또는 16바이트



4.3 정수형 데이터의 표현

C 언어의 정수형 데이터는 2진수로 표현되며, 부호가 있는 정수는 `2의 보수(Two’s Complement)` 방식으로 음수를 표현한다.

1. 부호 없는 정수

- `unsigned` 자료형은 양의 정수만 표현한다. 따라서, 같은 메모리 크기에서 보다 큰 양수를 저장할 수 있다.



2. 부호 있는 정수

- `signed` 자료형은 양수와 음수를 모두 표현할 수 있으며, `2의 보수` 방식으로 음수를 표현한다.

- 예시로 4비트에서 `1010`은 -6을 나타낸다. `2의 보수` 방식에서는 최상위 비트(MSB)가 1이면 음수를 의미한다.



4.4 부동 소수점과 연산

C 언어에서는 실수값을 표현하기 위해 부동 소수점(floating-point) 방식을 사용한다. 부동 소수점은 매우 큰 수나 매우 작은 수를 표현할 수 있어, 소수점이 포함된 계산에 유리하다.

1. 표현 방식

- `float`과 `double` 자료형을 사용해 실수값을 표현하며, 일반적으로 `IEEE 754` 표준을 따른다.

- 부동 소수점은 부호, 지수, 가수로 구성되어 있으며, `float`은 4바이트로 약 7자리, `double`은 8바이트로 약 15자리의 유효 자릿수를 표현한다.

2. 연산 오차와 제한사항

- 부동 소수점은 정확한 값을 표현하기 어려운 경우가 있으며, 특히 무한소수의 경우 오차가 발생할 수 있다.

- 오차는 소수점을 근사하여 표현하기 때문에 발생하며, `float`보다 `double`이 정밀도가 높다.

텍스트, 폰트, 스크린샷, 친필이(가) 표시된 사진

자동 생성된 설명

4.5 수식과 연산자

C 언어는 다양한 연산자를 제공하며, 각 연산자는 특정 작업을 수행하는 데 사용된다. 연산자는 크게 산술, 관계, 논리, 비트 연산자 등으로 나눌 수 있다.

1. 산술 연산자

- 사칙 연산을 수행하며, `+`, `-`, `\*`, `/`, `%` 등이 있다. 산술 연산자 중 `%`는 정수 간 나눗셈의 나머지를 구하는 연산자이다.



2. 관계 연산자

- 두 값을 비교하여 참(1) 또는 거짓(0)을 반환하는 연산자로, `==`, `!=`, `<`, `>`, `<=`, `>=` 등이 있다.

텍스트, 폰트, 친필, 라인이(가) 표시된 사진

자동 생성된 설명

3. 논리 연산자

- 논리식을 결합하거나 부정하여 논리적 참/거짓을 판단하는 연산자로, `&&`(AND), `||`(OR), `!`(NOT) 등이 있다.

텍스트, 폰트, 친필, 라인이(가) 표시된 사진

자동 생성된 설명

4. 비트 연산자

- 정수를 2진수로 간주하여 비트 단위로 연산을 수행하는 연산자로, `&`, `|`, `^`, `~`, `<<`, `>>` 등이 있다.

- 비트 연산자는 효율적인 메모리 사용과 특정 비트의 제어를 가능하게 한다.



4.6 연산자 우선순위와 결합 순서

연산자 우선순위는 여러 연산자가 하나의 수식에 포함될 때 계산 순서를 결정하며, 결합 순서는 동일한 우선순위의 연산자가 있을 경우 처리 방향을 의미한다.

1. 우선순위

- `\*`, `/`, `%` 등의 산술 연산자는 `+`, `-`보다 높은 우선순위를 가진다.

- 관계 연산자는 산술 연산자보다 낮고, 논리 연산자는 관계 연산자보다 낮다.

2. 결합 순서

- 대부분의 산술, 관계, 논리 연산자는 왼쪽에서 오른쪽으로 결합된다.

- 대입 연산자(`=`, `+=`, `-=`, 등)는 오른쪽에서 왼쪽으로 결합된다.



복잡한 수식을 작성할 때는 괄호를 사용하여 우선순위를 명시적으로 지정하는 것이 좋다. 이를 통해 코드의 가독성을 높이고, 의도한 대로 수식이 평가되도록 할 수 있다.

4.7 형변환 (Type Casting)

형변환은 서로 다른 자료형 간의 변환을 의미하며, C 언어에서는 자동 형변환과 명시적 형변환을 지원한다.

1. 자동 형변환 (Implicit Type Conversion)

- 더 높은 정밀도의 자료형으로 자동 변환이 이루어진다. 예를 들어, `int`와 `float`이 함께 사용되면 `int`가 `float`로 자동 변환된다.

2. 명시적 형변환 (Explicit Type Conversion)

- 개발자가 형변환을 명시적으로 지정하는 것으로, `(자료형)`을 사용하여 강제 변환을 수행한다.

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

형변환은 정밀도 손실을 방지하고, 자료형 간 연산에서 예상치 못한 오류를 방지하는 데 중요한 역할을 한다.

자료형과 연산은 프로그램에서 데이터를 표현하고 조작하는 데 필수적인 요소이다. 자료형을 선택하고 올바른 연산자를 활용하여 데이터의 의미와 정확성을 보장하는 것은 효율적인 코드 작성의 중요한 부분이다. 이를 이해하고 활용하는 것은 C 언어뿐만 아니라 모든 프로그래밍 언어에서 필수적인 기초이다.