C for begineer

Made by 서민상

1. 시작은 Hello World부터

c언어는 헤더 파일, main함수, 기타 함수 등으로 구성된다. 헤더 파일은 c언어를 사용할 때 쓰는 다양한 함수들이 저장된 파일이다. 출력, 입력 등의 간단한 함수들은 <stdio.h>파일에 들어있고, 그 밖에도 수학적인 계산으로 위해 쓰이는 <math.h>, 윈도우 제어를 위해 쓰이는 <window.h> 파일 등이 있다.
#헤더파일

Hello, World를 출력하기 위해선 <stdio.h> 헤더파일 안의 printf함수를 사용해야 한다. 사용법은 다음과 같다.

`printf(“쓸 문구”);`

이때 반드시 “” 안에 텍스트를 넣어야 한다.

최종적으로 문자를 출력하려면

```
#include <stdio.h>
int main(void)
{
    printf(“Hello, World!”);
    return0;
}
```

이렇게 하면 된다.

관련문서

- 학습시 사용할 교재
- 헤더 파일이란?

2-1 변수의 자료형

C언어에는 정수형, 실수형, 문자형의 총 3가지 자료형이 존재한다.
정수형은 6개의 범위가 존재한다.
1. **int** - 4바이트의 자료형이며, 음의 정수와 양의 정수를 표현가능하다.
2. **short int** - 2바이트의 자료형이며, int와 마찬가지로 음의 정수와 양의 정수를 담을 수 있지만, 그 범위가 2배 작다.
3. **unsigned int** -4바이트 크기의 자료형으로 int에서 음의 정수에 해당하는 범위를 지우고, 양의 정수를 담을 수 있는 범위를 2배 늘린 자료형이다.
4. **unsigned short int** - unsigned int와 특징이 비슷하나 자료형의 범위가 2바이트로 unsigned int보다 표현할 수 있는 범위가 2배 작다.
5. **long** - 8바이트의 자료형으로 int보다 담을 수 있는 정수의 범위가 2배 크나 64비트 운영체제에서는 4바이트의 크기를 가진다.
6. **unsigned long int** -4바이트의 자료형으로 unsigned int와 마찬가지로 양의 정수만을 표기가능하나 그 범위가 unsigned int의 2배이다.

c 언어에서 이렇게 사용한다.

```c
#include <stdio.h>
int main(void)
{
    int a = 4;
    short int b = 2;
    unsigned int c = 4;
    long d = 8;
    unsigned short int e = 4;
}
```

#int

다음은 float자료형에 관한 설명이다.
1. **float** - 소수를 표기할 수 있는 특징이 있는 자료형으로, 4바이트의 자료형이다.
2. **double** - 4바이트인 float 자료형에 비해 그 범위가 2배 넓으며, 8바이트의 크기를 가진 자료형이다.
3. **long double** - double 자료형의 확장된 버전으로 10바이트의 크기를 가진다.

c 언어에서 이렇게 사용한다.

```c
#include <stdio.h>
int main(void)
{
    float a = 4;
    double b = 8;
    long double c = 10;
}
```

#float

다음은 char 자료형에 관한 설명이다.
1. **char** - 아스키 코드에 있는 문자를 담을 수 있는 자료형으로 1바이트의 크기를 가진다.
2. **signed char** - 1바이트의 크기를 가지며 표현가능한 범위나 특징이 char 자료형과 같다.
3. **unsigned char** - 1바이트의 크기를 가지며 unsigned int 자료형과 마찬가지로 음의 정수는 표현하지 못하고, 양의 정수만 char 자료형의 양의 정수 부분이 표현할 수 있는 범위보다 2배 더 넓다.

c 언어에서 이렇게 사용한다.

```c
#include <stdio.h>
int main(void)
{
    char a = 1;
    signed char b = 1;
    unsigned char c = 1;
}
```

#char

END

#자료형

2-2 자료형의 크기 확인하기

이 노트는 2-1 변수의 자료형과 이어진다.

2-1 변수의 자료형에서 확인한 다양한 자료형들의 크기를 확인하기 위해선 `sizeof(A)`함수를 사용해야 한다.
다음은 각 자료형들의 크기를 출력하는 예시코드이다.

```c
#include <stdio.h>

int main() {
    int integer;
    short int shortInteger;
    unsigned int unsignedInteger;
    long longInteger;
    unsigned long unsignedLongInteger;
    unsigned short int unsignedShortInteger;
    float floatingPoint;
    double doublePrecision;
    long double longDoublePrecision;
    char character;
    signed char signedCharacter;
    unsigned char unsignedCharacter;

    printf("Size of int: %zu bytes\n", sizeof(integer));
    printf("Size of short int: %zu bytes\n",
sizeof(shortInteger));
    printf("Size of unsigned int: %zu bytes\n",
sizeof(unsignedInteger));
    printf("Size of long: %zu bytes\n", sizeof(longInteger));
    printf("Size of unsigned long int: %zu bytes\n",
    sizeof(unsignedLongInteger));
    printf("Size of unsigned short int: %zu bytes\n",
    sizeof(unsignedShortInteger));
    printf("Size of float: %zu bytes\n",
sizeof(floatingPoint));
    printf("Size of double: %zu bytes\n",
sizeof(doublePrecision));
    printf("Size of long double: %zu bytes\n",
sizeof(longDoublePrecision));
    printf("Size of char: %zu byte\n", sizeof(character));
    printf("Size of signed char: %zu byte\n",
sizeof(signedCharacter));
    printf("Size of unsigned char: %zu byte\n",
sizeof(unsignedCharacter));

    return 0;
}
```

이 코드의 결과는 다음과 같다.

```
Size of int: 4 bytes
Size of short int: 2 bytes
Size of unsigned int: 4 bytes
Size of long: 8 bytes
Size of unsigned long int: 8 bytes
Size of unsigned short int: 2 bytes
Size of float: 4 bytes
Size of double: 8 bytes
Size of long double: 16 bytes
Size of char: 1 byte
Size of signed char: 1 byte
Size of unsigned char: 1 byte
```

#sizeof #자료형

3-1. 숫자, 문자를 입력하려면?

1. 'scanf' 함수

C언어에서는 변수를 출력할 때 `scanf()`라는 입력 함수를 사용한다. `scanf`는 `printf`와 유사하게 입력 형식을 제어하는 부분과 입력 대상으로 구분된다. `scanf`를 사용하는 예시는 다음과 같다.

```
scanf("입력 형식 제어", &입력 대상(변수));
```

`scanf`로 정수, 실수, 문자를 입력하는 방법은 다음과 같다.

> 정수형 입력의 경우 : `scanf("%d", &a);` 단, 이때 a는 **int**형 변수이다.
> 실수형 입력의 경우 : `scanf("%f", &b);` 단, 이때 b는 **float**형 변수이다.
> 문자형 입력의 경우 : `scanf("%c", &c);` 단, 이때 c는 **char**형 변수이다.

2-1. 형식지정자

입력 형식을 제어하는 부분의 %d, %f와 같은 부분은 #형식지정자 로 입력하려는 변수에 따라 다르게 사용한다.

정수형 형식지정자의 경우 다음과 같다.

> 1. `%d` - 10진 정수형 데이터의 입력과 출력에 사용된다.
> 2. `%o` - 8진 정수형 데이터의 입력과 출력에 사용된다.
> 3. `%x` - 16진 정수형 데이터의 입력과 출력에 사용된다.

실수형 형식지정자의 경우 다음과 같다.

> 1. `%f` - 실수형 데이터의 입력과 출력에 사용된다.

문자형 형식지정자의 경우 다음과 같다.

> 1. `%c` - 문자형 데이터의 입력과 출력에 사용된다.

형식지정자의 경우 형식지정자 문서에서 더 자세하게 설명한다.

2-2. '&'의 의미

`scanf`함수의 '&'는 입출력을 관리하는 포인터로 추후 포인터문서에서 자세하게 설명한다.
'&'뒤에오는 문자는 앞서 선언한 변수이며, 이 변수에 `scanf`함수를 통해 문자나 숫자가 할당되게 되고, `printf`함수를 통해 할당된 변수가 출력되게 된다.
다음은 `scanf`함수를 통해 할당된 숫자를 `printf`함수를 통해 출력하는 예시이다.

```c
#include <stdio.h>
int main(void)]
{
    int a;
    scanf("%d", &a);
    printf("%d", a);

    return 0;
}
```

이때 `printf`로 출력하려는 변수의 앞에는 '&'를 붙이지 않는다.

이 코드의 출력 결과는 다음과 같다.

```
//a에 3을 입력할 경우

<cd~/coding> 3
```

3-2. 형식지정자에 관하여

`printf`나 `scanf`와 같은 함수를 사용하기 위해 #형식지정자 가 꼭 필요하다. 형식지정자는 각 문자형에 따라 또는 출력하는 변수의 속성에 따라 문자형, 정수형, 실수형으로 구분된다.

문자형, 정수형, 실수형 형식지정자의 경우 3-1. 숫자, 문자를 입력하려면?에 작성되어있기에 이 문서는 그 세부사항을 다루고 있다.

정수형 형식지정자의 경우 다음과 같은 세부 형식지정자들이 있다.

1. `%u` - unsigned int형 변수에 관한 형식지정자이다.
2. `%lu` - unsigned long형 변수에 관한 형식지정자이다.
3. `%lld` - long long형 변수에 관한 형식지정자이다.
4. `%ld` - long형 변수애 관한 형식지정자이다.

실수형 형식지정자의 경우 다음과 같은 세부 형식지정자들이 존재한다.

1. `%Ld` - long double형 변수에 관한 형식지정자이다.
2. `%lf` - double형 변수에 관한 형식지정자이다.
3. `%0.2f`, `%0.5f`등 - 실수형 변수를 수소점 몇번째자리까지 나타내는지를 지정하는 형식지정자이다.

문자형 형식지정자의 경우 다음과 같은 세부 형식지정자들이 존재한다.

1. `%c` - 문자형 변수에 관한 형식지정자이다.
2. `%su` - 유니코드 문자열 변수에 관한 형식지정자이다.
3. `%s8` - UTF-8형 변수에 관한 형식지정자이다.
4. `%ma` - ASCII 64개의 변수에 관한 형식지정자이다.

이번 문서에 언급한 형식지정자 말고도 더 다양한 형식지정자들이 있다.
더 자세한 형식지정자에 관한 정보를 얻고 싶다면 여기를 클릭하면 된다.

3-3. 아스키 코드

C언어에서 문자를 입력하기 위해선 문자형 형식지정자에서 보았듯이 `%c`를 사용해야 한다.
그런데 컴퓨터는 어떤 원리로 우리가 입력한 문자를 저장하는걸까?
바로 **아스키 코드**를 이용하는 것이다.

아스키 코드란 문자를 **1대 1**로 대응시킨 숫자이다. 우리가 **scanf**를 통해 입력한 문자는 컴파일 시 약속된 정수값인 아스키 코드값에 대응하는 숫자로 변환된다.

> 따라서 결국 문자또한 **메모리에 저장되는 정수이다.**

아스키 코드표

아스키 코드표는 다음과 같다.

"Pasted image 20230720105732.png" could not be found.

> **1. 숫자 10개 `>>` 48~57**
> **2. 영어 대문자 26개 `>>` 65~90**
> **3. 영어 소문자 26개 `>>` 97~122**
> **4. 특수문자 33개 `>>` 32 ~**
> **5. 제어문자 33개 `>>` 0, 9, 10, 13**

이때 제어문자란 `\0`, `\n`, `\t`와 같이 특별한 의미를 부여하여 출력을 제어하기 위한 문자를 의미한다.

참고문서

- 숫자 또는 문자를 입력하려면
- 문자형 형식지정자

4-1. if문

C언어에는 조건이 충족될 시 괄호 안의 명령을 실행하는 조건문이 존재한다.
조건문은 `#if` 문 `#while` 문의 2가지가 존재한다.

if문은 ()안의 조건, {}안의 실행명령 2가지 부분으로 구성된다.
이때 ()안의 조건에 대한 내용은 관계 연산자에 상세히 설명되어 있다.
()안에는 비교, 등치, 부등식, 형식 확인 등 다양한 조건을 할당할 수 있고, {}안에는 기존 C언어 문법을 거의 다 사용할 수 있다. ~~if문 안에 if문을 사용할 수 있다.~~

다음은 if문을 사용하는 간단한 예시이다.

```c
#include <stdio.h>
int main(void)
{
    int a, b;
    a = 1, b = 1;

    if(a == b)
    {
        printf("a equals to b");
    }
    return 0;
}
```

등치에 사용하는 연산자는 '='가 아닌 '=='를 사용한다.
그 이유는 `a = b`의 의미는 a 라는 변수에 b라는 값을 할당하라는 의미이기 때문이다.
따라서 a 와 b를 비교하기 위해선 수학적으로 '='와 동일한 의미를 가지는 '==' 연산자를 사용해야 한다.

참고문서

- 변수의 자료형
- 형식지정자
- 관계 연산자

4-2. if문에서의 AND, OR연산

C언어를 사용하다 보면, 하나의 조건이 아닌 두가지 조건을 동시에 만족하거나 두 가지 조건 중 한가지를 만족하는 경우 값이 출력되어야 하는 경우가 있다.

=> **이때 사용하는 것이 AND, OR연산자이다.**

AND연산자는 `&&`를 사용하고, OR연산자는 `||`연산자를 사용한다.

AND연산자를 사용하는 대표적인 예시

> 1. 입력한 변수가 3보다 크고 5보다 작을 경우 `Correct!`를 출력하는 경우
> 2. a가 정수이고, b가 정수일 경우 a와 b의 합을 출력하는 경우
> 3. a가 'True'이고 b가 5보다 클 경우 5까지의 자연수의 합을 출력하는 경우

OR연산자를 사용하는 대표적인 예시

> 1. a>3 또는 b<3를 만족할 때 두 수의 합을 출력하는 경우
> 2. y>0 또는 ax<0일 때 이 함수의 그래프를 출력하는 경우

예제

1. AND연산자를 사용하는 예시코드

```c
#include <stdio.h>
int main(void)
{
    int a=1, b=5;

    if(a<b && b>0)
    {
        printf("Done!")
    }
    return 0;
}
```

2. OR연산자를 사용하는 예시코드

```c
#include <stdio.h>
int main(void)
{
    int a=5, b;
    scanf("%d", &b)

    if(a<b || b>0)
    {
        printf("Hello!");
    }
    return 0;
}
```

참고문서

- if문
- scanf
- 변수의 자료

5-1. while문

#if 문과 마찬가지로 특정조건이 충족될 경우 {}안의 내용을 출력하는 조건문이다. 그러나 #if 문과의
차이점은 명백하다. #if 문은 조건을 충족한 뒤 {}안의 내용을 실행하면 괄호 밖으로 빠져나오지만 #while 문은 조건이 충족되는 한 계속 반복으로 {}안의 내용을 실행한다.

#while **문의 구성은 다음과 같다.**

```
while(조건문)
{
        실행문
}
```

다음은 while문을 실행하는 예시코드이다.

```c
#include <stdio.h>
int main(void)
{
    int i;
    while(i<100)
    {
        printf("The Number is %d", i);
        i++;
    }
    return 0;
}
```

참고문서

- if문
- 숫자 또는 문자를 입력하려면
- 변수의 자료형
- 정수형 형식지정자

6-1 증가, 감소 연산자

1. 증가, 감소 연산자에 대한 개념

1-1. 개념

`++`와 `--` 연산자는 각각 증가, 감소 연산자라 부른다. 대부분의 연산자들은 `a+b`, `a*b`와 같이 두 개의 피연산자를 필요로 하는 **2항(binary)연산자**이지만, 이 연산자들의 경우 피연산자가 1개 존재할 경우 사용할 수 있는 **단항(unary)연산자**이다.

1-2. 사용법

증가, 감소 연산자는 변수의 앞 또는 뒤에 사용되어 그 변수에 저장된 값을 1 증가 또는 1 감소시킨다.
Ex) >> `++(a+b)`, `3++`, `(x-y)--` 등의 방법으로 사용할 수 있다.

2. 전행 연산자와 후행 연산자

2-1 개념

증가 연산자(++)나 감소 연산자(--)가 변수의 앞에 위치한 경우를 **전행 연산자**, 변수의 뒤쪽에 위치한 경우를 **후행 연산자**라고 한다.

2-2 무엇이 다른가

계산 결과에 있어 차이가 발생하게 된다. 이는 다음의 예제를 통해 알아볼 수 있다.

```c
#include <stdio.h>
int main(void)
{
    int a, b;
    a = 10;
    b = 20;

    printf("\n a++\'s result is: %d", a++);
    printf("\n ++a\'s result is: %d", ++a);
    printf("\n b--\'s result is: %d", b--);
    printf("\n --b\'s result is: %d", --b);

    return 0;
}
```

<실행 결과>

식	결과
a++	10
++a	12
b--	20
--b	18

결과에서 알 수 있듯이 2의 차이가 발생한다. 이러한 현상의 원인은 증가, 감소 연산자의 작동 방식에서 찾을 수 있다.
선행 연산에서는 변수의 값을 **우선으로 증가**시킨 다음 변수에 대한 평가(출력, 사칙연산...etc)를 진행하고,
후행 연산에서는 변수의 값을 **먼저 평가**한 후 변수의 값을 변화시키는 작동방식을 가진다.

`a++`에서 a는 후행 연산자에 의해서 값이 1 증가하게 되는데, 이때 a값에 대한 평가(print)가 먼저 발생한 뒤 a의 값을 1 증가시켜 a는 11이 되고, a++의 결과는 10으로 출력된다.
`++a`에서 a는 선행 연산자에 의해서 값이 1 증가하게 되는데, 이때 a의 값이 먼저 증가되게 되고(11+1) 이후 변수에 대한 평가가 이루어지므로 a의 값은 12가 되고, ++a를 출력한 결과는 12가 된다.
b의 경우도 이와 같이 일어난다.

for문에서의 활용

뒤에 등장할 내용인 for문에서 #for 문의 구조를 보면 알 수 있지만, for문은 제어변수를 증가시켜 원하는 횟수까지 반복시키는 구조로 이루어져 있다. 이때 제어변수를 증가시키는 증감식에서 주로
증가, 감소 연산자를 사용한다.
다음은 for문에서 증가, 감소 연산자를 이용하는 예시이다.

```c
#include <stdio.h>
int main(void)
{
    for(int i=0; i<5; i++)
    {
        printf("%d\n", i);
    }
    return 0;
}
```

위 코드를 보면 한 번의 반복마다 증가 연산자를 통해 제어변수의 값을 1 증가시키는 것을 볼 수 있다.

참고문서

- for문
- 산술 연산자

6-2. for문

#while 문과 마찬가지로 ()안의 조건(주로 범위에 관한 내용)을 만족할 경우 {}안의 내용을 계속 출력하는 함수이다. ()안은 총 3가지(2가지)부분으로 구성되며, 각 부분들은 다음과 같다.

> 1. 변수 초기화

for문을 몇 번 반복할지를 결정하는 변수(주로 i를 사용한다)의 초기값을 설정한다.

> 2. 범위 설정

사용된 변수가 얼마만큼 증가하는지를 설정한다. (ex> i>3 && i<5)

> 3. 증가 or 감소? (생략 가능)

순환에 사용되는 변수를 매 순환마다 증가시킬지, 혹은 감소시킬지를 결정한다.(ex> i++)

각 부분들은 `;`를 사용하여 구분한다.
다음은 위 세 부분들을 결합한 #for 문의 예시이다.

```c
#inlcude <stdio.h>
int main(void)
{
    for(int i=0; i<100; i++)
    {
        printf("%d\n", i);
    }
    return 0;
}
```

참고문서

- while문
- if문
- 정수형 형식지정자
- 증가 & 감소 연산자

7. switch, case문

한 가지 변수로 여러 개의 #if 문을 작성하려면 어떻게 해야할까?
물론 이런 방식으로 할 수 있다.

```c
int a=5;

if(a==5)
{
    printf("5");
}

else if(a<5)
{
    printf("a<5");
}

else if(a>5)
{
    printf("a>5");
}
```

그러나 이러한 방식은 코드를 작성하는 시간도 많아 걸리고 가독성도 떨어진다. 그런데. C언어에는 이러한 상황을 해결하기 위해 #switch 문이라는 것이 존재한다. `switch`문은 한 가지 변수에 대해 많은 상황을
가정할 때 용이한 함수로 `switch(a)`로 변수를 할당하고 `case '조건문' : 실행문; break;`로 사용할
수 있다. 특이한 점은 #break 가 사용된다는 점인데, 한 가지 `case`를 만족하고 실행문을 실행하면 **코드진행은 계속 그곳에 머물러 있게 된다.** 따라서 `break`함수를 사용해서 그 코드에서 빠져나와야 한다.

다음은 `switch`문을 사용한 간단한 예제코드이다.

```c
#include <stdio.h>
int main(void)
{
    char s;
    scanf("%c", &s);

    switch(s)
        case 'W' : printf("WINTER"); break;
        case 'S' : printf("SPRING"); break;
        case 'SU' : printf("SUMMER"); break;
        case 'F' : printf("FALL"); break;

    return 0;
}
```

참고문서

- 숫자 또는 문자를 입력하려면?
- 문자형 형식지정자
- if문

8-1. 함수의 기본 개념

함수란?

1-1. 개념

함수란 특별한 기능을 수행하도록 만들어진 프로그램의 단위를 의미한다. 특히 프로그래밍에서의 함수란 **재사용 가능한 코드 블록을 생성하는 데 사용되는 구성 요소**를 의미한다.
일반적으로 C언어를 포함한 프로그래밍 언어에서의 함수는 다음과 같은 형태를 가진다.

```c
[반환 유형] 함수이름([매개변수 리스트]) {
    // 함수의 코드 블록
    // 코드 블록에서 필요한 작업 수행
    // 필요한 경우 반환문을 사용하여 결과 반환
}
```

1-2. 그렇다면 이때까지 사용한 것들의 정체는?

물론 함수이다. 'Hello world!'를 출력하기 위해 사용했던 `printf()`도, 변수에 특정한 값을 입력하기 위해 사용했던 `scanf("%d", &a)`도 물론 함수이다. `if문`, `while문`이 함수인 것은 말할 것도 없다. 이때까지 사용한 모든 것은 **다** 함수였다.

1-3. 더 많은 함수?

그렇다면 이때까지 등장한 함수는 C언어가 가진 함수의 얼마 정도 될까? 아마 단 **5%** 도 되지 않을 것이다.
아직 C언어를 사용하기 위해 필요한 최소한의 헤더 파일인 <stdio.h>안의 함수도 다 등장하지 못했음에도
불구하고 C언어에는 <stdio.h>를 포함한 수많은 헤더 파일이 존재하고, 각 헤더 파일에는
또 수많은 함수가 들어있다. 심지어 이 헤더 파일은 사용자들이 직접 만들 수 있기 때문에, 이론상 C언어가 보유한 함수는 셀수 없다.

참고문서

- 시작은 Hello World 부터
- 헤더 파일
- 숫자 또는 문자를 입력하려면
- if문
- while문

- for문

8-2. 헤더 파일

헤더 파일이란?

라이브러리 함수

C언어에는 프로그래머들이 사용할 수 있도록 '미리 만들어진' 함수들이 있다.
`printf()`, `scanf()`와 같은 함수들이 그 예시이다. 이 함수들은 **헤더 파일**인 <stdio.h>에 포함되어 있었고, 그것이 매번 이러한 함수들을 사용하기 위해 `#include`를 통해 <stdio.h>파일을 불러와야 했던
이유이다.

헤더 파일

헤더 파일은 **.h**라는 확장자를 사용하는 파일로, 특정한 함수와 그 함수에 대한 정의가 포함되어 있다.
헤더파일들은 컴파일러가 설치된 **include** 폴더에 들어있으며, 특정한 라이브러리 함수를 사용하기 위해 선
이 헤더파일을 `#include`를 통해 import해야 한다.

다음은 C언어에서 사용되는 대표적인 헤더 파일들을 표로 정리한 것이다.

헤더 파일	함수들의 기능	포함된 함수
stdio.h	표준 입출력	printf, scanf...
ctype.h	문자 처리	isalpha, tolower...
math.h	수학적인 계산(log 등)	sin, sqrt, log...
stdlib.h	메모리 관리를 포함한 함수들	rand, srand...
string.h	문자열 처리	strlen, strcpy...
time.h	날자와 시간	time, acstime

참고문서

- 함수의 기본 개념
- 시작은 Hello World 부터

8-4. 수학 계산을 하는 다양한 함수들

이전 문서인 <math.h>를 알아보자에서 표를 통해 알아보았던 수학 계산 함수들 중
몇 가지를 더 알아볼 것이다.

1. 절댓값 계산 함수

이전 문서의 표에 등장했던 함수인 `int abs(int a)`를 예제를 통해 더 자세히 알아볼 것이다.
다음은 **abs**함수를 이용해 입력한 정수의 절댓값을 구하는 간단한 예시이다.

```c
#include <stdio.h>
#include <math.h>

int main(void)
{
    int a;
    printf("Enter a number: ");
    scanf("%d", &a);
    printf("\n %d\'s absolute value is: %d", a, abs(a));

    return 0;
}
```

2. 거듭제곱의 계산

다음은 이전 문서에서 거듭제곱을 계산하는 함수로 등장한 **pow**함수를 예제를 통해 알아볼 것이다.
다음은 **pow**함수를 이용해 두 정수의 x^y의 값을 구하는 간단한 예시이다.

```c
#include <stdio.h>
#include <math.h>

int main(void)
{
    int x, y;
    int x = 10;
    int y = 3;

    printf(”pow(10, 3) = %d\n”, pow(10, 3));

    return 0;
}
```

<실행결과>

```c
pow(10, 3) = 1000
```

3. 양의 제곱근을 계산하는 함수

다음은 이전 문서에서 x의 양의 제곱근을 계산하는 함수로 등장한 **sqrt**함수를 예제를 통해 알아볼 것이다.
다음은 **sqrt**함수를 이용해 정수 x의 양의 제곱근을 구하는 간단한 예시이다.

```c
#include <stdio.h>
#include <math.h>

int main(void)
{
    printf(“sqrt(4) = %d\n”, sqrt(4));
    printf(“sqrt(2) = %.3f\n”, sqrt(2));

    return 0;
}
```

<실행결과>

```c
sqrt(4) = 2
sqrt(2) = 1.414
```

참고문서

- <math.h>를 알아보자
- 헤더 파일
- 함수의 기본 개념

8.5 rand() 함수를 이용한 난수 생성

C언어로 수학적인 계산을 하려면?

1-1. math.h

헤더 파일 편에서 알아본 것처럼 C언어에는 수많은 헤더 파일과, 그 안의 수많은 함수들이 존재한다.
이중 C언어에서 수학적인 계산(log, sin, √, |x| ...etc)을 하기 위해 필요한 헤더파일은 **math.h**이다.

1-2. math.h 안의 다양한 수학 계산 함수들

math.h안에는 절댓값, 지수함수, 삼각함수를 계산하는 데 필요한 함수들 뿐만이 아닌 다양한
수학적 계산을 위한 함수들이 포함되어 있다. 다음은 이중 자주 쓰이는 함수들을 표로 정리한 것이다.

유형	함수 원형	설명
절댓값	int abs(int a)	int 형의 절댓값을 계산한다
절댓값	double fabs(double b)	double 형의 절댓값을 계산한다
지수함수	int pow(int x, int y)	x의 y제곱의 값을 계산한다
지수함수	int exp(int a)	자연상수 e(약 2.7182)의 'a'제곱의 값을 계산한다
로그함수	int log(int x)	log(x)의 값을 계산한다
로그함수	int log10(int y)	밑이 10인 y의 로그를 계산한다
제곱근	int sqrt(int x)	x의 양의 제곱근을 계산한다
sin함수	int sin(int a)	sin(a)를 계산한다
cos함수	int cos(int b)	cos(b)를 계산한다
tan함수	int tan(int c)	tan(c)를 계산한다
올림	double ceil(double x)	주어진 숫자보다 크거나 같은 최소 정수 값을 구한다
내림	double floor(double x)	주어진 숫자보다 작거나 같은 최대 정수 값을 구한다

1-3. 함수 원형

함수 원형(prototype)이란 함수의 자료형, 사용할 인자, 함수의 이름을 정의한 부분을 말한다.
프로그래밍을 할 때 변수를 사용하기 위해서 변수의 자료형과 이름을 선언하듯이(예를 들어, **int a=10;**)
함수 역시 자료형을 요구한다. 아래는 위 표의 **pow 함수**를 예시로 함수 원형을 설명한 것이다.

int pow(int x, int y)

인자:

- `x`: 밑(base)으로 사용되는 정수 값
- `y`: 지수(exponent)로 사용되는 정수 값

반환 값:

`x`의 `y` 제곱을 계산한 결과를 정수 형태로 반환한다.

설명:

`pow` 함수는 주어진 `x` 값의 `y` 제곱을 계산하여 그 결과를 정수 형태로 반환한다.
즉, `x^y`를 계산하여 결과를 반환한다.

다음은 **pow 함수**를 사용하는 간단한 예시이다.

```c
#include <stdio.h>
#include <math.h>

int main(void)
{
    int result = pow(2, 3);
    printf("2^3 = %d\n", result);

    return 0;
}
```

출력결과

```
2^3 = 8
```

참고문서

- 헤더 파일

비트 연산자

비트연산자에 대한 개념

1-1. 개념

컴퓨터 안에서 처리되는 모든 데이터들은 비트(bit)형태로 표현되고 저장된다.
이때 비트연산자(bit operator)는 이 데이터들을 비트 단위로 연산한다.
비트 연산자는 비트 논리 연산자와 비트 시프트 연산자로 구분되는데, 자세한 내용은 아래의 표와 같다.

<비트 논리 연산자>

연산자	사용방법	의미
~	~a	a에 저장된 값에 대한 1의 보수
&	a&b	a에 저장된 값과, b에 저장된 값의 비트 단위의 논리곱
|	a | b	a에 저장된 값과, b에 저장된 값의 비트 단위의 논리합

처음 보는 단어가 3개나 등장한다.

1. 이때 **보수**는 수학적인 개념이므로 여기를 참조하면 좋다.

2. **논리곱**은 (**AND**)라고도 하며, 주어진 두 조건이 모두 참인 경우에만 전체 식을 참으로 평가한다.
   수학 집합에서의 교집합을 생각하면 이해가 쉬울 것이다.
   논리곱은 (**OR**)라고도 하며 **if문**의 AND, OR연산에도 사용되지만, 이번 비트 논리 연산에서는 두 수를 비트 단위로 비교한다.

다음은 **논리곱**을 사용하는 간단한 예시이다.

```c
0 & 0 -> 결과: 0
0 & 1 -> 결과: 0
1 & 0 -> 결과: 0
1 & 1 -> 결과: 1
```

3. **논리합**은 (**OR**)라고도 하며, 주어진 두 조건중 하나가 참인 경우 전체 식을 참으로 평가한다.
   수학 집합에서의 합집합과 대응되는 개념이라고 생각하면 이해가 쉬울 것이다.
   논라합은 논리곱과 마찬가지로 **if문**의 조건식에 사용되지만, 이번 비트 논리 연산에서는 두 수를 비트 단위로 비교하여 연산한다.

다음은 논리합을 사용하는 간단한 예시이다.

```c
1 | 0 -> 결과: 1
0 | 1 -> 결과: 1
1 | 1 -> 결과: 1
0 | 0 -> 결과: 0
```

<비트 시프트 연산자>

연산자	사용방법	의미
^	a^b	a의 값과 b의 값에 대한 베타적 논리합
<<	a<<b	a의 값을 b만큼 왼쪽으로 시프트 연산
>>	a>>b	a의 값을 b만큼 오른쪽으로 시프트 연산

역시 모르는 단어가 3개 등장한다.

1. **베타적 논리합** 역시 수학에서의 개념이므로 여기를 참조하는것이 도움이 될 것이다.
   베타적 논리합은 (**XOR**)라고도 하며, 두 조건(여기서는 비트)이 서로 다른 경우에만 참(1)의 값을 가진다.
   벤 다이어그램으로 나타낼 경우 아래의 그림처럼 표현된다.

"Pasted image 20230806163726.png" could not be found.

다음은 **베타적 논리합** 연산자를 사용하는 간단한 예시이다.

```c
0 ^ 0 -> 결과: 0
0 ^ 1 -> 결과: 1
1 ^ 0 -> 결과: 1
1 ^ 1 -> 결과: 0
```

2. 각 시프트 연산자(`>>`, `<<`)는 **2항 연산자**로써 정수를 어느 자릿수로 이동시킬지 결정하는 연산자이기에 **부호 없는 정수형 데이터**(unsigned int)자료형을 사용한다.

<<(오른쪽 시프트)연산자는 비트열을 오른쪽으로 이동하면서 왼쪽 비트를 모두 0으로 채운다.
>>(왼쪽 시프트)연산자는 비트열을 왼쪽으로 이동하면서 오른쪽 비트를 모두 0으로 채운다.

다음은 각 비트 시프트 연산을 그림으로 표현한 것이다.

<왼쪽 비트시프트 연산의 경우>

"Pasted image 20230806164759.png" could not be found.

<오른쪽 비트시프트 연산의 경우>

"Pasted image 20230806164917.png" could not be found.

> 비트 시프트의 개념이 생소할 수 있지만 결국, 오른쪽 비트 시프트 연산자는 **2로 나눈 효과**를, 왼쪽 비트 시프트 연산자는 **2를 곱한 효과**를 낸다.

다음은 비트 시프트 연산을 사용한 간단한 예시이다.

<오른쪽 비트시프트 연산>

```c
#include <stdio.h>

int main() {
    int num = 16; // 00010000 (2진수)

    printf("오른쪽 시프트 이전: %d (2진수: %d)\n", num, num);

    num = num >> 1;

    printf("오른쪽 시프트 이후: %d (2진수: %d)\n", num, num);

    return 0;
}
```

<Result>

```
오른쪽 시프트 이전: 16 (2진수: 10000)
오른쪽 시프트 이후: 8 (2진수: 1000)
```

<왼쪽 비트시프트 연산>

```c
#include <stdio.h>

int main() {
    int num = 4; // 00000100 (2진수)

    printf("왼쪽 시프트 이전: %d (2진수: %d)\n", num, num);

    num = num << 2;

    printf("왼쪽 시프트 이후: %d (2진수: %d)\n", num, num);

    return 0;
}
```

<Result>

```
왼쪽 시프트 이전: 4 (2진수: 100)
왼쪽 시프트 이후: 16 (2진수: 10000)
```

참고문서

- 조건 연산자
- 관계 연산자
- 관계 연산자
- 산술 연산자

산술 연산자

1. 사칙연산에 필요한 산술 연산자

C언어로 사칙연산을 하기 위해서는 산술 연산자가 필요하다. 산술연산자는 총 5가지가 존재하며,
변수간에 사칙연산을 진행할 때 사용된다. 다음은 산술 연산자들의 종류와 그 쓰임을 표로 정리한 것이다.

연산자	사용방법	의미
+	a+b	a에 저장된 값과 b에 저장된 값을 서로 더한다
-	a-b	a에 저장된 값과 b에 저장된 값을 서로 공제한다
-	-a	a에 저장된 값의 부호를 반대로 바꾼댜
*	a*b	a에 저장될 값과 b에 저장된 값을 서로 곱한다
/	a/b	a에 저장된 값을 b에 저장된 값으로 나눈다
%	a%b	a에 저장된 값을 b로 나눈 값의 나머지

다음은 이 연산자들을 사용하여 간단한 사칙연산을 수행하는 코드이다.

```c
#include <stdio.h>
int main(void)
{
    double x, y;
    x = 2.5;
    y = 5.0;
    printf("\n x+y = %.3f", x+y);
    printf("\n x-y = %.3f", x-y);
    printf("\n x*y = %.3f", x*y);
    printf("\n x/y = %.3f", x/y);
    printf("\n x%%y = %.3f", x%y);

    return 0;
}
```

다음은 이 코드의 실행 결과이다.

식	결과
x+y	7.5

식	결과
x-y	-2.5
x*y	12.5
x/y	0.5
x%y	-

%연산자와 /연산자에 대한 비교

다음은 혼동하기 쉬운 `%`연산자와 `/`연산자에 대한 비교이다.

</ 연산자>

식	결과
12/4	3
16/3	5
5/2	2.5

<%연산자>

식	결과
12%4	3
16%3	1
5%2	-

> 이때 %연산자의 3번째 결과가 `-`인 이유는 피연산자로 실수형은 사용할 수 없기 때문이다.

참고문서

- if문을 사용한 계산기 만들기
- 변수의 자료형

조건 연산자

조건 연산자에 대한 개념

1-1. 개념

조건 연산자(conditional operator)는 3개의 피연산자를 사용하는 3항 연산자이다. `?`와 `:`를 사용하며
수식 1의 결과가 참 혹은 거짓일 경우에 따라 `:` 앞뒤로 구분되는 2개의 수식 중 어느 것을 실행시킬지가 달라지게 된다. **if문을 한 줄로 작성하는 경우**라고 생각해도 된다.

1-2. 사용법

1. `condition1 ? condition2 : condition3`

   > condition1의 연산 결과가 참이라면 **condition2**를 실행,
   > 거짓일 경우 **conditon3**을 실행하라는 의미다.

2. `X = (a>b) ? a : b;`

   > a가 b보다 클 경우 **a의 값을 X에 저장**, b가 a보다 클 경우 **b의 값을 X에 저장**하라는 의미이다.

다음은 조건 연산자를 사용하는 간단한 예시 코드이다.

```c
#include <stdio.h>

int main() {
    int num1, num2, max;

    // 사용자로부터 두 개의 정수 입력받기
    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);

    // 조건 연산자를 사용하여 더 큰 값을 max에 저장
    max = (num1 > num2) ? num1 : num2;

    // 더 큰 값을 출력
    printf("더 큰 값은: %d\n", max);

    return 0;
}
```

참고문서

- 관계 연산자
- 증가 & 감소 연산자
- 산술 연산자
- if문

관계 연산자

1. 관계 연산자에 대한 개념

1-1. 개념

관계 연산자(relational operator)는 데이터 간의 대소 관계를 비교하는데 사용하며, 다음의 표와 같다.

연산자	사용방법	의미	결과값
<	a<b	a에 저장된 값과 b에 저장된 값을 서로 더한다	참이면 1, 거짓이면 0
<=	a<=b	a에 저장된 값과 b에 저장된 값을 서로 공제한다	참이면 1, 거짓이면 0
>	a>b	a에 저장된 값의 부호를 반대로 바꾼댜	참이면 1, 거짓이면 0
>=	a>=b	a에 저장될 값과 b에 저장된 값을 서로 곱한다	참이면 1, 거짓이면 0
==	a==b	a에 저장된 값을 b에 저장된 값으로 나눈다	참이면 1, 거짓이면 0
!=	a!=b	a에 저장된 값을 b로 나눈 값의 나머지	참이면 1, 거짓이면 0

1-2. 사용법

뒤의 문서에 등장할 `#if` 문과 같이 변수가 어떠한 범위를 만족시킬 때 특정 함수를 실행하는 경우, 변수들의 크기를 비교해서 그 결과를 출력하는 경우에 주로 사용된다.

다음은 관계 연산자를 사용하는 간단한 예시이다.

```c
#include <stdio.h>
int main(void)
{
    int a, b;
    a = 5;
    b = 10;

    printf("%d\n", a>b);
    printf("%d\n", a>=b);
    printf("%d\n", a<b);
    printf("%d\n", a<=b);
    printf("%d\n", a==b);
    printf("%d\n", a!=b);

    return 0;
}
```

If문에서의 활용

if문은 조건이 충족될 시 괄호{} 안의 함수를 실행하는 구조로 되어있는데, 이때 변수의 범위에 따라 특정 함수를 실행하기 위해 관계 연산자가 주로 사용된다.

다음은 if문에서 관계연산자를 사용하는 간단한 예시이다.

```c
#include <stdio.h>
int main(void)
{
    int a;

    printf("Enter a number: ");
    scanf("%d", &a);

    if(a>0)
    {
        printf("a is a postive number!");
    }
    return 0;
}
```

참고문서

- if문
- 6-1 증가, 감소 연산자
- 산술 연산자

for문을 이용한 구구단 출력

#for 문을 사용하여 구구단을 출력하는 예시이다.

> 1. `for` 문은 총 2번 사용한다.
> 2. 첫 번째 `for`문은 구구단의 첫 번째 곱하는 수를 1씩 증가시킬 것이다.
> 3. 두 번째 `for`문은 구구단의 두 번째 곱하는 수를 1씩 증가시킴과 동시에 구구단을 출력한다.
> 4. 첫 번째 `for`문의 끝에 `printf("\n");`을 추가하여 각 단 사이를 구분한다.

다음은 `for`문을 이용하여 구구단을 출력하는 예시이다.

```c
#include <stdio.h>
int main(void)
{
    int i, j;

    for(i=2; i<10; i++)
    {
        for(i=j; j<10; j++)
        {
            printf("%d * %d = %d", i, j, i*j);
        }
        printf("\n");
    }
    printf("\n");
    return 0;
}
```

참고문서

- for문
- 정수형 형식지정자
- 변수의 자료형

3차원 행렬 출력하기

I. 3차원 행렬이란?

C언어에는 행렬을 추가할 수 있는 기능이 존재한다. 행렬은 int값을 가지며
int (변수이름)[]...[];으로 나타낸다. 이때 []의 갯수에 따라 행렬의 차원이 달라진다.

ex1) 3차원 행렬

```c
#include <stdio.h>
int main(void)
{
    int matrix = {
        {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9},
        },
        {
            {10, 11, 12},
            {13, 14, 15},
            {16, 17, 18},
        },
        {
            {19, 20, 21},
            {22, 23, 24},
            {25, 26, 27},
        },
    };
}
```

다음은 3차원 행렬을 완전히 출력하는 예시이다.

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

    int i, j, k;
    int matrix = {
        {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9},
        },
        {
            {10, 11, 12},
            {13, 14, 15},
            {16, 17, 18},
        },
        {
            {19, 20, 21},
            {22, 23, 24},
            {25, 26, 27},
        },
    };


    for(i=0; i<3; i++)
    {
        for(j=0; j<3; j++)
        {
            for(k=0; k<3; k++)
            {
                printf("%d ", matrix[i][j][k]);
            }
            printf("\n");
        }
        printf("\n");
    }
    return 0;
}
```

위 예제를 보면 for문을 3중으로 겹쳐 3차원 행렬을 완전히 출력하는 것을 볼 수 있다. 첫 번째 for문은 행렬들의 세 묶음을 선택하여 출력, 그 다음 for문은 행렬들의 묶음 안의 하위 행렬 세 묶음 중 하나를 선택하

여 출력, 마지막 for문은 행렬 안의 상분 3개를 선택하여 출력한다.

관련문서

- 변수의 자료형
- 형식지정자
- 입출력변수

if문을 사용한 계산기 만들기

#if 문을 사용하여 계산기를 만드는 과정이다.

> 1. 두 정수의 덧셈을 계산한다고 가정, 2개의 변수를 담을 #int 형 변수 2개와, 연산기호를 담을 #char 형 변수를 선언한다.
> 2. #if 문의 조건을 연산기호에 따라 설정한 뒤 각 연산기호에 맞는 문구를 출력하도록 코딩한다.
> 3. 컴파일 후 실행한다.

다음은 #if 문을 사용하여 2개의 정수에 관하 사칙연산을 수행하는 코드이다.

```c
#include <stdio.h>
int main(void)
{
    int a, b;
    char operand;

    printf("두 정수를 입력하시오: ");
    scanf("%d, %d\n", &a, &b);
    printf("연산기호를 입력하시오: ");
    scanf("%c", &operand);

    if(operand == '+')
    {
        printf("%d + %d = %d", a, b, a+b);
    }
    else if(operand == '-')
    {
        printf("%d - %d = %d",a, b, a-b);
    }
    else if(operand == '*')
    {
        printf("%d * %d = %d", a,  b, a*b);
    }
    else if(operand == '/')
    {
        printf("%d / %d = %d", a, b, a/b);
    }
    else
    {
        printf("Invaild operand! choose the right operand");
    }
    return 0;
}
```

a = 4, b =2, operand = '+'로 설정하고 출력한 결과는 다음과 같다.

```c
4 + 2 = 6

done
```

참고문서

- 변수의 자료형
- 형식지정자
- 숫자 또는 문자를 입력하려면?
- if문