# 프로그래밍 1
# Lecture Note #03

백윤철
ybaek@smu.ac.kr

# 목차

- 변수
- 변수의 자료형
- 산술연산자
- 증가 감소 연산자
- 관계연산자
- 연산자 우선순위
- 데이터 입력
- C의 keyword

# 덧셈 프로그램 구현에 필요한 + 연산자

```
int main(void)
{
    3 + 4; /* 3과 4의 합을 계산할 것을 명령함 */
    return 0;
}
```

실행결과로는 아무것도
나타나지 않습니다.

- 컴파일 및 실행 시 문제가 발생하지 않음. 따라서 코드의 문제는 없음
- + 기호는 숫자의 덧셈을 의미함. 따라서 실행하면 3과 4의 합을 계산함
- +같은 기호를 연산자라고 부름

# 덧셈 프로그램 구현에 필요한 + 연산자

- 연산의 결과는?
  - + 연산만 요구하였지, 그 결과를 출력하기 위한 어떠한 코드도 삽입되지 않음
  - 따라서 출력되는 것이 없음
  - 연산의 결과를 저장해야 원하는 바를 추가로 진행할 수 있음
  - 연산결과 또는 값을 저장할 수 있도록 C언어에서는 변수(variable)이라는 것을 제공함

# 변수를 이용한 데이터의 저장

- 변수란?
  - 값을 저장할 수 있는 메모리 공간에 붙여진 이름
  - 변수라는 것을 선언하면 메모리 공간이 할당되고 할당된 메모리 공간에 이름이 붙는다.

- 변수의 이름
  - 변수의 이름을 통해서 할당된 메모리 공간에 접근이 가능하다.
  - 값을 저장할 수도 있고 저장된 값을 참조할 수도 있다.

# 변수를 이용한 데이터의 저장

```
int main(void)
{
    int num;
    num = 20;
    printf("%d", num);
    …
}
```

```
int num;
```

· int—정수의 저장을 위한 메모리 공간 할당

· num - 할당된 메모리 공간의 이름은 num

```
num = 20;
```

· 변수 num이 위치한 메모리 공간에 20을 저장

```
printf("%d", num);
```

· num에 저장된 값을 참조(사용)해서 출력

# 변수의 다양한 선언 및 초기화 방법

```
int main(void)
{
    int num1, num2; // 변수 num1, num2 정의
    int num3 = 30, num4 = 40; // 정의 및 초기화
    printf("num1: %d, num2: %d\n", num1, num2);
    num1 = 10; // 변수 num1 초기화
    num2 = 20; // 변수 num2 초기화
    printf("num1: %d, num2: %d\n", num1, num2);
    printf("num3: %d, num4: %d\n", num3, num4);
    return 0;
}
```

실행결과

```
num1: -858993460, num2: -858993460
num1: 10, num2: 20
num3: 30, num4: 40
```

# 변수의 다양한 선언 및 초기화 방법

```
int num1, num2;
```

- 변수를 정의만 할 수 있음
- 콤마를 이용하여 둘 이상의 변수를 동시에 정의할 수 있음
- 변수를 정의만 하면 값이 대입되기 전까지 쓰레기 값(의미 없는 값)이 채워짐

```
int num3 = 30, num4 = 40;
```

- 변수의 정의와 동시에 초기화 하는 것이 가능함

# 변수 정의할 때 주의할 사항

```
int main(void)
{
    int num1;
    int num2;
    num1 = 0;
    num2 = 0;

    …
}
```

컴파일 가능한 변수 선언

```
int main(void)
{
    int num1;
    num1 = 0;
    int num2;
    num2 = 0;

    …
}
```

컴파일 불가능한 변수 선언

- 과거의 C 표준(ANSI C)에서는 변수의 선언이 실행 코드 시작 전에 있어야 했음
  - 지금도 그 표준을 따르는 컴파일러가 존재함

# 식별자(변수, 함수 이름 등)의 이름 규칙

- 변수의 이름은 알파벳, 숫자, 언더바(_)로 구성됨
- 변수의 이름은 알파벳이나 언더바로 시작해야 함
- C언어는 대소문자를 구별함
  - 변수 Num과 num은 다른 변수임
- C언어의 키워드는 변수의 이름으로 사용할 수 없음
- 이름 사이에 공백이  들어갈 수 없음

**괜찮은 경우**

```
int Th7Val;
int phone;
int your_name;
```

**잘못된 경우**

```
int 7ThVal;     /* 숫자로 시작함 */
int phone#;     /* 특수문자 사용 */
int your name; /* 공백문자 사용 */
```

# 변수의 자료형 (Data Type)

- 숫자형 두 가지 부류
  - 정수형 변수와 실수형 변수
- 정수형 변수
  - 정수 값을 저장하기 위해 만들어진 변수
  - char, short, int, long 형 변수로 나누어짐
- 실수형 변수
  - 실수 값(소수점이 있는 값)을 저장하기 위해 만들어짐
  - float과 double 형 변수로 나누어짐
- 정수형 변수와 실수형 변수가 나누어지는 이유?
  - 정수와 실수를 저장하는 방식이 다르기 때문

# 변수의 자료형 (Data Type)

```
int num1 = 24
```

- num1은 정수형 변수 중 int형 변수

```
double num2 = 3.14
```

- num2는 실수형 변수 중 double 형 변수

```
int main(void) {
    int num1 = 3;
    int num2 = 4;
    int result = num1 + num2;
    printf("덧셈 결과: %d\n", result);
    printf("%d + %d = %d\n", num1, num2, result);
    printf("%d와 %d의 합은 %d\n", num1, num2, result);
    return 0;
}
```

실행결과

```
덧셈 결과: 7
3+4=7
3와(과) 4의 합은 7입니다.
```

**변수를 선언하여 덧셈의 결과를 저장했기 때문에 덧셈결과를 다양한 형태로 출력할 수 있다.**

# 대입 연산자와 산술 연산자

| 연산자 | 설명 | 결합방향 |
|---|---|---|
| = | 연산자 오른쪽에 있는 값을 연산자 왼쪽에 있는 변수에 대입. 예) num = 20; | ← |
| + | 두 피연산자의 값을 더함. 예) num = 4 + 3; | → |
| - | 왼쪽의 피연산자 값에서 오른쪽의 피연산자 값을 뺌. 예) num = 4 - 3; | → |
| * | 두 피연산자의 값을 곱함. 예) num = 4 * 3; | → |
| / | 왼쪽의 피연산자 값을 오른쪽의 피연산자 값으로 나눔. 정수와 실수를 구분함.<br>예) num = 7 / 3; | → |
| % | 왼쪽의 피연산자 값을 오른쪽의 피연산자 값으로 나눌 때 얻게 되는 나머지 값을 반환. 정수 연산에서만 사용됨<br>예) num = 7 % 3; | → |

# 대입 연산자와 산술 연산자

```
int main(void) {
    int num1 = 9, num2 = 2;
    printf(%d+%d=%d\n", num1, num2, num1+num2);
    printf(%d-%d=%d\n", num1, num2, num1-num2);
    printf(%d×%d=%d\n", num1, num2, num1*num2);
    printf(%d÷%d=%d\n", num1, num2, num1/num2);
    printf(%d÷%d의 나머지=%d\n", num1, num2,
            num1%num2);
    return 0;
}
```

실행결과

```
9+2=11
9-2=7
9×2=18
9÷2의 몫=4
9÷2의 나머지=1
```

**함수호출 문장에 연산식이 있는 경우 연산이 이뤄지고 그 결과를 기반으로 함수의 호출이 진행된다.**

# 복합 대입 연산자

- 동일 연산

| 산술 연산 | 복합 대입 연산자 활용 |
|---|---|
| a = a + b | a += b |
| a = a - b | a -= b |
| a = a * b | a *= b |
| a = a / b | a /= b |
| a = a % b | a %= b |

# 복합 대입 연산자

```
int main(void)
{
    int num1 = 2, num2 = 4, num3 = 6;
    num1 += 3; /* num1 = num1 + 3; */
    num2 *= 4; /* num2 = num2 * 4; */
    num3 %= 5; /* num3 = num3 % 5; */
    printf("Result: %d, %d, %d\n",
            num1, num2, num3);
    return 0;
}
```

실행결과

```
Result: 5, 16, 1
```

# 부호의 의미를 갖는 +연산자와 - 연산자

```
int main(void)
{
    int num1 = +2; /* int num1 = 2; 와 같음 */
    int num2 = -4;
    num1 = -num1;
    printf("num1: %d\n", num1);
    num2 = -num2;
    printf("num2: %d\n", num2);
    num1 = +num1;
    printf("num1: %d\n", num1);
    num2 = +num2;
    printf("num2: %d\n", num2);
    return 0;
}
```

실행결과?

# 부호의 의미를 갖는 +연산자와 - 연산자

- 두 연산자를 혼동하지 않도록 주의

```
num1=-num2;    /* 부호 연산자 사용 */
num1-=num2;    /* 복합 대입 연산자 사용 */
```

- 띄어쓰기를 통해 혼동을 최소화시킴

```
num1 = -num2; /* 부호 연산자 사용 */
num1 -= num2; /* 복합 대입 연산자 사용 */
```

# 증가, 감소 연산자

| 연산자 | 연산자의 기능 | 결합방향 |
|---|---|---|
| ++num | 값을 1 증가 후, 속한 문장의 나머지 연산을 진행 (선 증가, 후 연산)<br>예) val = ++num; | ← |
| num++ | 속한 문장의 연산을 먼저 진행한 후, 값을 1 증가 (선 연산, 후 증가)<br>예) val = num++; | ← |
| --num | 값을 1 감소 후, 속한 문장의 나머지 연산을 진행 (선 감소, 후 연산)<br>예) val = --num; | ← |
| num-- | 속한 문장의 연산을 먼저 진행한 후, 값을 1 감소 (선 연산, 후 감소)<br>예) val = num--; | ← |

# 증가, 감소 연산자

```
int main(void)
{
    int num1 = 12;
    int num2 = 12;
    printf("num1: %d\n", num1);
    /* 후위 증가 */
    printf("num1++: %d\n", num1++);
    printf("num1: %d\n\n", num1);
    printf("num2: %d\n", num2);
    /* 전위 증가 */
    printf("++num2: %d\n", ++num2);
    printf("num2: %d\n", num2);
    return 0;
}
```

실행결과

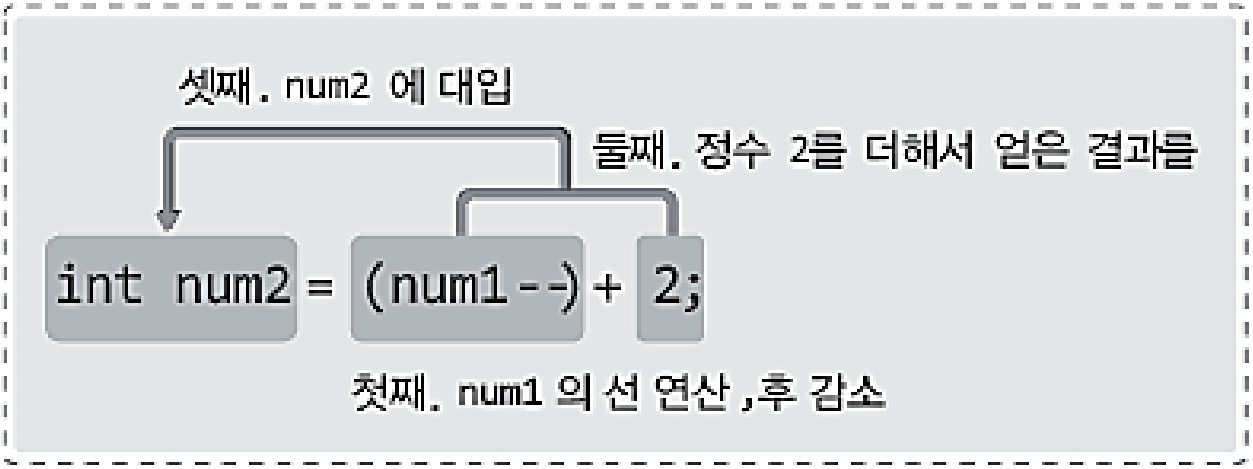
```
num1: 12
num1++: 12
num1: 13

num2: 12
++num2: 13
num2: 13
```

# 증가, 감소 연산자 추가 예제

```
int main(void)
{
    int num1 = 10;
    int num2 = (num1--) + 2;
    printf("num1: %d\n", num1);
    printf("num2: %d\n", num2);
    return 0;
}
```

실행결과

```
num1: 9
num2: 12
```




**연산의 과정**

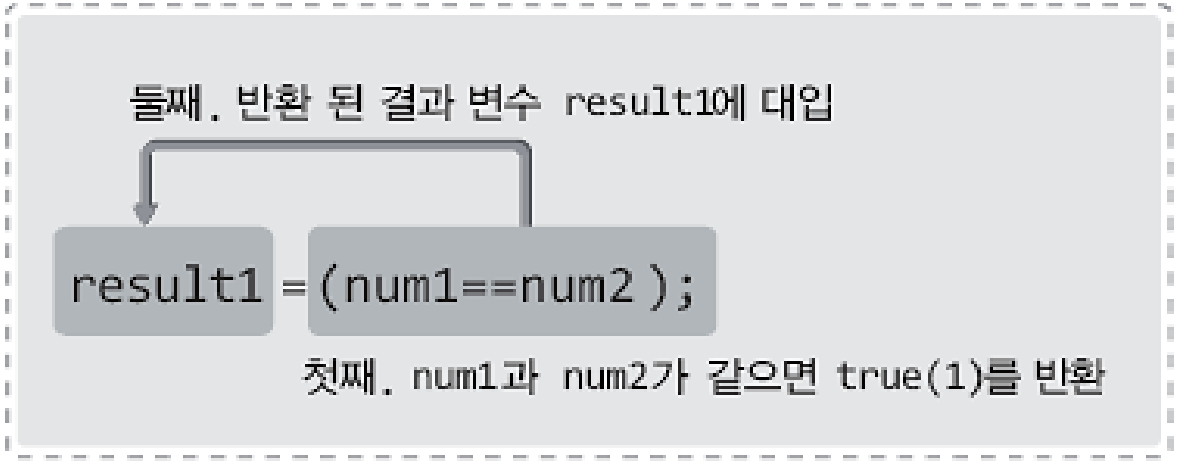
# 관계 연산자

| 연산자 | 연산자의 기능 | 결합방향 |
|---|---|---|
| < | 예) n1 < n2<br>n1이 n2보다 작은가? | → |
| > | 예) n1 > n2<br>n1이 n2보다 큰가? | → |
| == | 예) n1 == n2<br>n1과 n2가 같은가? | → |
| != | 예) n1 != n2<br>n1과 n2가 다른가? | → |
| <= | 예) n1 <= n2<br>n1이 n2보다 같거나 작은가? | → |
| >= | 예) n1 >= n2<br>n1이 n2보다 같거나 큰가? | → |

# 관계 연산자

- 관계 연산자
  - 연산의 조건을 만족하면 참을 의미하는 1을 반환하고 만족하지 않으면 거짓을 의미하는 0을 반환하는 연산자
    - C언어에서는 0이 아닌 모든 값이 참
    - 1은 참을 의미하는 대표적인 값일 뿐, 꼭 1이 참을 의미하는 것은 아님

# 관계 연산자




```
int main(void)
{
    int num1 = 10;
    int num2 = 12;
    int result1, result2, result3;
    result1 = (num1 == num2);
    result2 = (num1 <= num2);
    result3 = (num1 > num2);
    printf("result1: %d\n", result1);
    printf("result2: %d\n", result2);
    printf("result3: %d\n", result3);
    return 0;
}
```

실행결과

```
result1: 0
result2: 1
result3: 0
```

# 콤마 연산자

- 콤마 (,) 연산자
  - 콤마도 연산자로 취급
  - 둘 이상의 변수를 동시에 선언하거나 둘 이상의 문장을 한 행에 삽입하는 경우에 사용되는 연산자
  - 둘 이상의 인자를 함수로 전달할 때 인자의 구분을 목적으로 사용됨
  - 콤마 연산자는 다른 연산자들과 달리 연산의 결과가 아닌 '구분'을 목적으로 함
  - 콤마 연산자는 가장 오른쪽에 있는 값을 결과로 반환함
  - 주로 for 반복문에서 사용 (개념적으로 모호할 수 있음)

# 콤마 연산자

```
#include <stdio.h>
int main() {
    int a[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    int b[20] = { 0, };
    int i, j, k;
    for (i = 0, j = 10; i < 10; i++, j++) {
        b[j] = a[i];
    }
    for (i = 0; i < 20; i++)
        printf("%d\n", b[i]);
    return 0;
}
```

# 연산자의 우선순위와 결합방향

- 연산자의 우선순위
  - 연산의 순서에 대한 정의
  - 덧셈과 뺄셈보다는 곱셈과 나눗셈의 우선순위가 높음
- 연산자의 결합방향
  - 우선순위가 동일한 두 연산자 사이에서의 연산 진행 방향
  - 덧셈, 뺄셈, 곱셈, 나눗셈 모두 결합방향이 왼쪽에서 오른쪽으로 진행

```
3 + 4 × 5 ÷ 2 - 10
```

  - 연산자의 우선순위에 근거해서 곱셈과 나눗셈이 먼저 진행됨
  - 결합방향에 근거하여 곱셈이 나눗셈보다 먼저 진행됨

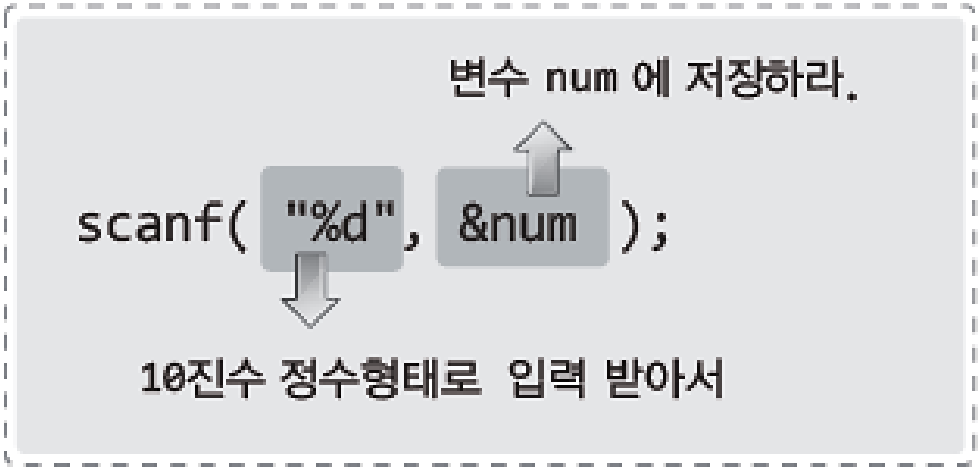
# 연산자 우선순위

- 연산자 우선순위

| 순위 | 연산기호 | 연산자 | 결합방향 |
|---|---|---|---|
| 1위 | () | 함수호출 | → |
| | [] | 인덱스 | |
| | −〉 | 간접지정 | |
| | . | 직접지정 | |
| | ++ (postfix)<br>−− (postfix) | 후위증가 및 감소 | |
| 2위 | ++ (prefix)<br>−− (prefix) | 전위증가 및 감소 | ← |
| | sizeof | 바이트 단위 크기 계산 | |
| | ~ | 비트 단위 NOT | |
| | ! | 논리 NOT | |
| | −, + | 부호 연산(음수와 양수의 표현) | |
| | & | 주소 연산 | |
| | * | 간접지정 연산 | |
| 3위 | (casting) | 자료형 변환 | ← |
| 4위 | *, /, % | 곱셈, 나눗셈 관련 연산 | → |
| 5위 | +, − | 덧셈, 뺄셈 | → |
| 6위 | 〈〈, 〉〉 | 비트이동 | → |
| 7위 | 〈, 〉, 〈=, 〉= | 대소비교 | → |
| 8위 | ==, != | 동등비교 | → |
| 9위 | & | 비트 AND | → |
| 10위 | ^ | 비트 XOR | → |
| 11위 | \| | 비트 OR | → |
| 12위 | && | 논리 AND | → |
| 13위 | \|\| | 논리 OR | → |
| 14위 | ? : | 조건연산 | ← |
| 15위 | =, +=, −=, *=, /=, %=,<br>〈〈=, 〉〉=, &=, ^=, \|= | 대입연산 | ← |
| 16위 | , | 콤마연산 | → |

# 키보드를 이용해서 정수 입력을 받기 위한 함수

```
int main(void)
{
    int num;
    scanf("%d", &num);
    …
}
```




- printf()함수에서의 %d는 10진수 정수의 출력을 의미
- scanf()함수에서의 %d는 10진수 정수의 입력을 의미함
- 변수의 이름 num 앞에 붙는 &는 'address of 연산자' 나중에 설명함 (당장은 붙여야 하는 걸로만 이해)

# 키보드를 이용해서 정수 입력을 받기 위한 함수

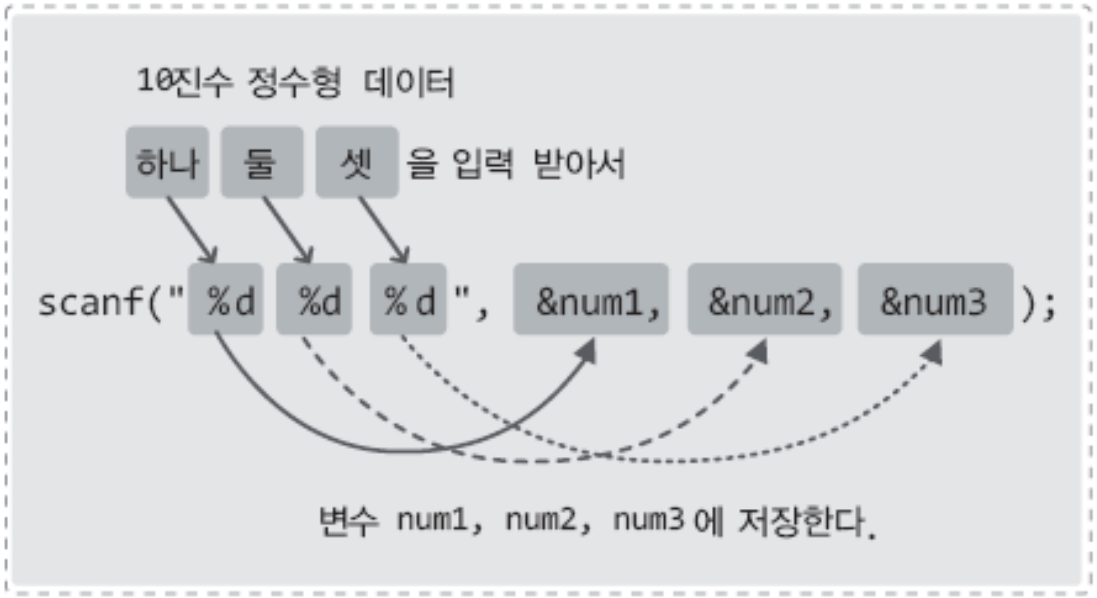
```c
#include <stdio.h>
int main(void) {
    int result;
    int num1, num2;
    printf("정수 one: ");
    scanf("%d", &num1);
    printf("정수 two: ");
    scanf("%d", &num2);
    result = num1 + num2;
    printf("%d + %d = %d\n", num1, num2, result);
    return 0;
}
```

실행결과

```
정수 one: 3
정수 two: 4
3 + 4 = 7
```

# 입력의 형태를 다양하게 지정 가능

```
int main(void)
{
    int num1, num2, num3;
    scanf("%d %d %d",
          &num1, &num2, &num3);
    …
}
```




- 한 번의 scanf 함수 호출을 통해서 둘 이상의 데이터를 원하는 방식으로 입력 받을 수 있음

```c
#include <stdio.h>
int main(void) {
    int result;
    int num1, num2, num3;
    printf("세 개의 정수 입력: ");
    scanf("%d %d %d", &num1, &num2, &num3);
    result = num1 + num2 + num3;
    printf("%d + %d + %d = %d\n",
            num1, num2, num3, result);
    return 0;
}
```

실행결과

```
세 개의 정수 입력: 4 5 6
4 + 5 + 6 = 15
```

# C언어의 표준 키워드

- 키워드
  - C언어의 문법을 구성하는 그 의미가 결정되어 있는 단어

| | | | |
|---|---|---|---|
| auto | _Bool | break | case |
| char | _Complex | const | continue |
| default | do | double | else |
| enum | extern | float | for |
| goto | if | _Imaginary | return |
| restrict | short | signed | sizeof |
| static | struct | switch | typedef |
| union | unsigned | void | volatile |
| while | | | |

# 정리

- 변수
- 변수의 자료형
- 산술연산자
- 증가 감소 연산자
- 관계연산자
- 연산자 우선순위
- 데이터 입력
- C의 keyword