Kumoh National Institute of Technology

C Programming

Ch.1-3 변수와 연산자

컴퓨터공학과

C프로그래밍

Contents

1. 연산을 위한 연산자와 값의 저장을 위한 변수
2. C 언어의 다양한 연산자 소개
3. 키보드로 데이터 입력하기
4. C 언어 키워드
5. scanf() Vs. scanf_s() (tip)

- tip : Visual C++ 사용 방법 등 참고 자료

덧셈 연산자

- 연산자란 무엇인가?
  - 연산을 요구할 때 사용되는 기호
  - 예) +, -, *, /
- 연산자의 적용한 형태 → 수식
  - 수식의 결과로 항상 값이 나오게 됨 // 평가하다.
  - 예) 수식 (3 + 4)의 평가값은 7

```
int main(void)
{
    3 + 4;          // 덧셈 결과를 저장할 필요가 있음!
    return 0;
}
```

변수!

변수를 이용한 데이터의 저장

- 변수란 무엇인가?
  - 데이터를 저장할 수 있는 메모리 공간에 붙여진 이름
  - 변수의 이름을 통해 할당된 메모리 공간에 접근 가능
- 다양한 형태(자료형)의 변수
  - 정수형 : char, int, long 등
  - 실수형 : float, double 등
- 변수의 선언 및 대입
  - 대입 연산자(=) : 값을 대입하기 위한 용도의 연산자

```
int main(void)
{
    int val;                // int형 변수 val의 선언
    val = 20;               // 변수 val에 20을 저장
    printf("%d", val);      // val 변수의 값을 읽어 출력
}
```

main 함수에서 return 문이 생략되면 자동으로 0이 반환됨

변수의 다양한 선언 및 초기화 방법

```
#include <stdio.h>

int main(void)
{
    int num1, num2;              // 쓰레기 값으로 초기화
    int num3 = 30, num4 = 40;  // 선언과 동시에 초기화 가능

    printf("num1 : %d, num2 : %d\n", num1, num2);
    num1 = 10;
    num2 = 20;

    printf("num1 : %d, num2 : %d\n", num1, num2);
    printf("num3 : %d, num4 : %d\n", num3, num4);
}
```

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

잠깐만!
실행 시 num1, num2가 초기화되지 않은 상태에서 사용되었다는 경고 메시지가 출력될 수 있음 → 무시를 클릭하고 계속 실행
현재 컴파일러(VS2022)에서는 에러(허용하지 않음)

변수 선언 시 주의 사항 (1)

- 변수 선언 위치
  - 변수를 사용하기 이전에 선언
  - 어느 위치든 선언 가능
    - cf) 옛날 표준 C : 항상 함수(블록)의 시작 위치에 선언

```
void main(void)
{
    int num1;
    num1 = 10;

    int num2;                    //  Ok!
    num2 = 10;
    ......
}
```

변수 선언 시 주의 사항 (2)

- 변수의 이름 규칙
  - 변수의 이름은 알파벳, 숫자, 언더바(_)로 구성됨
  - 대소문자를 구분함
    - Num과 num은 다른 변수!
  - 변수의 이름은 숫자로 시작 불가
  - C 언어의 키워드를 변수로 사용 불가
  - 공백이 포함될 수 없음
- 잘못된 변수 이름의 예

가능하면 의미있는 이름으로 작성!
no_of_students, max_value

```
int 7ThVal;       // 변수의 이름이 숫자로 시작했으므로
int phone#;       // 변수의 이름에 #과 같은 특수문자는 올 수 없다.
int your name;    // 변수의 이름에는 공백이 올 수 없다.
```

변수의 자료형

- 기본 자료형 두 가지
  - 정수형 변수, 실수형 변수
- 정수형 변수
  - 정수값 저장
  - char, short(short int), int, long(long int)
- 실수형 변수
  - 실수값 저장
  - float, double, long double
- 왜 이렇게 종류가 많지?
  - 정수와 실수의 저장 방식이 다름
  - 미리 정해 두면 프로그램의 실행 속도 향상
    - cf) python : 사용하기 편하지만 실행 속도가 느릴 수 있음

```
void main(void)
{
    int num1 = 24;
    double num2 = 3.14;
}
```

정수와 실수의 저장 방식은
4장에서 자세히 설명함

덧셈 프로그램의 완성

```
#include <stdio.h>

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(과) %d의 합은 %d입니다.\n", num1, num2, result);
}
```

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

대입 연산자와 산술 연산자 (1)

연산자	연산자의 기능	결합방향
=	연산자 오른쪽에 있는 값을 연산자 왼쪽에 있는 변수에 대입한다. 예) num = 20;	←
+	두 피연산자의 값을 더한다. 예) num = 4 + 3;	→
-	왼쪽의 피연산자 값에서 오른쪽의 피연산자 값을 뺀다. 예) num = 4 - 3;	→
*	두 피연산자의 값을 곱한다. 예) num = 4 * 3;	→
/	왼쪽의 피연산자 값을 오른쪽의 피연산자 값으로 나눈다. 예) num = 7 / 3;	→
%	왼쪽의 피연산자 값을 오른쪽의 피연산자 값으로 나눴을 때 얻게 되는 나머지를 반환한다. 예) num = 7 % 3;	→

- num1 = num2 = num3 = 100; 의 실행 순서
  - 오른쪽에서 왼쪽으로 진행

대입 연산자와 산술 연산자 (2)

```
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÷%d의 몫 = %d \n", num1, num2, num1 / num2);
    printf("%d÷%d의 나머지 = %d \n", num1, num2, num1 % num2);
}
```

함수호출 문장에 연산식이 있는 경우 연산이 이뤄지고 그 결과를 기반으로 함수의 호출이 진행됨

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

복합 대입 연산자

- 복합 대입 연산자
  - 대입 연산자와 산술 연산자가 합해진 다양한 형태의 대입 연산자
- 다음 프로그램의 실행 결과는?


```
int main(void)
{
    int num1 = 2, num2 = 4;
    int num3 = 6;
    num1 += 3;
    num2 *= 4;
    num3 %= 5;

    printf("Result: %d, %d, %d \n", num1, num2, num3);
}
```

부호 연산자로서의 +, -

```
int main(void)
{
    int num1 = +2;
    int num2 = -4;

    num1 = -num1;
    printf("num1: %d \n", num1);
    num2 = -num2;
    printf("num2: %d \n", num2);
}
```

```
num1: -2
num2: 4
```

-= 복합 대입 연산자와 =- 혼동하기 없기
num1 -= num2;
num1 =- num2; → num1 = -num2;

증가 연산자, 감소 연산자 (1)

- 변수의 값을 1 증가, 1 감소시키는 연산자
  - 최종적으로 반환되는 값에 주의!
  - 변수 1개에만 적용 가능
    - 3++ (X), (a + b)++ (X)

전위 증가 연산자 : ++num
후위 증가 연산자 : num—
전위 감소 연산자 : --num
후위 감소 연산자 : num--

연산자	연산자의 기능	결합방향
++num	값을 1 증가 후, 속한 문장의 나머지를 진행(선 증가, 후 연산) 예) val = ++num;	←
num++	속한 문장을 먼저 진행한 후, 값을 1 증가(선 연산, 후 증가) 예) val = num++;	→
--num	값을 1 감소 후, 속한 문장의 나머지를 진행(선 감소, 후 연산) 예) val = --num;	←
num--	속한 문장을 먼저 진행한 후, 값을 1 감소(선 연산, 후 감소) 예) val = num--;	→

증가 연산자, 감소 연산자 (2)

```
int main(void)
{
    int num1 = 12;
    int num2 = 12;

    printf("num1: %d \n", num1);
    printf("num1++: %d \n", num1++);
    printf("num1: %d \n\n", num1);

    printf("num2: %d \n", num2);
    printf("++num2: %d \n", ++num2);
    printf("num2: %d \n", num2);
}
```

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

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

* float, double 형 실수 변수 값에도 적용 가능
  float a = 3.1;
  a++; → 4.1
* 다음 모두 불가능 : 연산 결과가 변수가 아닌 값임!
  ++(++a), (++a)++, ++(a++), (a++)++

증가 연산자, 감소 연산자 (3)

- 다음 프로그램의 실행 결과는?

```
int main(void)
{
    int num1 = 10;
    int num2 = (num1--) + 2;

    printf("num1: %d \n", num1);
    printf("num2: %d \n", num2);
}
```

증가 연산자와 감소 연산자를 복잡하게 사용하지 말자!

```
int num1 = 10;
int num2 = num1++ + ++num1;
```

관계 연산자 (1)

- 관계 연산자 //비교 연산자라 부르기도 함
  - 연산의 조건을 만족하면 참을 의미하는 1을 반환하고 만족하지 않으면 거짓을 의미하는 0을 반환하는 연산자

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

* C 언어 : 명시적인 true, false 값과 자료형이 없음
  - 정수값 1과 0으로 표현
  - 0이 아닌 모든 값은 참으로 간주

관계 연산자 (2)

```
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);
}
```

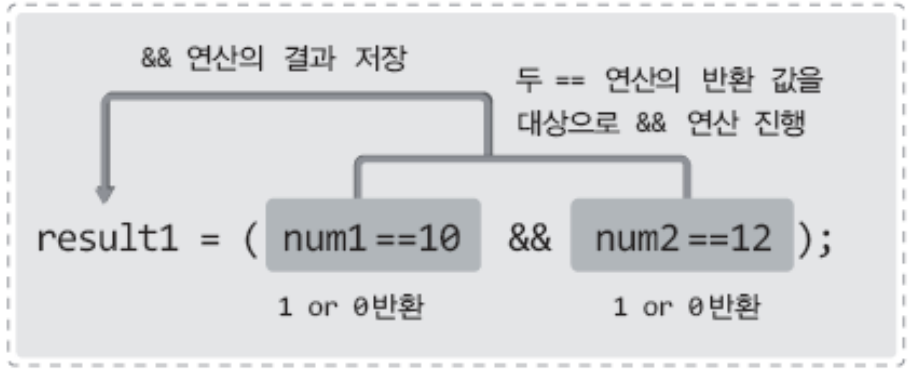
```
result1: 0
result2: 1
result3: 0
```

논리 연산자 (1)

- 논리곱(AND), 논리합(OR), 논리부정(NOT)

연산자	연산자의 기능	결합방향
&&	예) A && B A와 B 모두 '참'이면 연산결과로 '참'을 반환(논리 AND)	→
||	예) A || B A와 B 둘 중 하나라도 '참'이면 연산결과로 '참'을 반환(논리 OR)	→
!	예) !A A가 '참'이면 '거짓', A가 '거짓'이면 '참'을 반환(논리 NOT)	←

논리 연산자 (2)




```
int main(void)
{
    int num1 = 10;
    int num2 = 12;
    int result1, result2, result3;

    result1 = (num1 == 10 && num2 == 12);
    result2 = (num1 < 12 || num2 > 12);
    result3 = (!num1);

    printf("result1: %d \n", result1);
    printf("result2: %d \n", result2);
    printf("result3: %d \n", result3);
}
```

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

콤마 연산자

- 콤마 (,)
  - 콤마도 연산자임
  - 둘 이상의 변수를 동시에 선언하거나 둘 이상의 문장을 한 행에 삽입하는 경우에 사용
  - 둘 이상의 인자를 함수로 전달할 때 인자의 구분
  - 콤마 연산자는 다른 연산자들과 달리 연산의 결과가 아닌 '구분'을 목적으로 사용

```
int main(void)
{
    int num1 = 1, num2 = 2;

    printf("Hello "), printf("world! \n");
    num1++, num2++;
    printf("%d ", num1), printf("%d ", num2), printf("\n");
}
```

```
Hello world!
2 3
```

연산자의 우선순위와 결합방향 (1)

- 연산자의 우선순위
  - 연산의 실행 순서에 대한 순위
  - 덧셈과 뺄셈보다는 곱셈과 나눗셈의 우선순위가 높음
- 연산자의 결합방향
  - 우선순위가 동일한 두 연산자 사이에서의 연산을 실행하는 방향
  - 덧셈, 뺄셈, 곱셈, 나눗셈 모두 결합방향이 왼쪽에서 오른쪽으로 진행됨

```
3 + 4 * 5 / 2 - 10
```

- 우선순위에 따라 *, / 연산이 먼저 실행됨
- *, / 중에서는 결합방향에 따라 * 연산이 먼저 실행됨
- 우선순위와 결합방향을 잘 모르겠다면 → 괄호()를 사용하라!

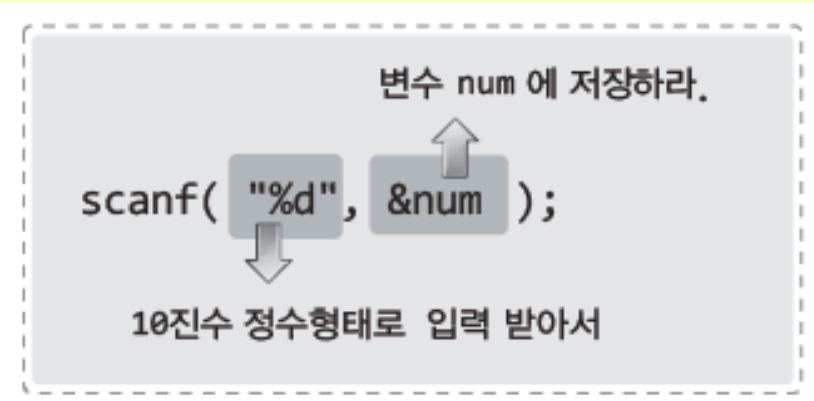
연산자의 우선순위와 결합방향 (2)

순위	기호	연산자	결합방향
1위	()	함수호출	→
	[]	인덱스	
	->	간접지정	
	.	직접지정	
	++, --	후위증가 및 감소	
2위	++, --	전위증가 및 감소	←
	sizeof	바이트 단위 크기 계산	
	~	비트 단위 NOT	
	!	논리 단위 NOT	
	-, +	부호 연산(음수와 양수의 표현)	
	&	주소연산	
	*	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	| (Shift + ₩)	비트 OR	→
12위	&&	논리 AND	→
13위	|| (Shift + ₩ * 2)	논리 OR	→
14위	? :	조건 연산	←
15위	=, +=, -=, *=, /=, %=, <<=, >>=, &=, |=	대입 연산	←
16위	,	콤마 연산	→

앞으로 하나씩 배우게 됨

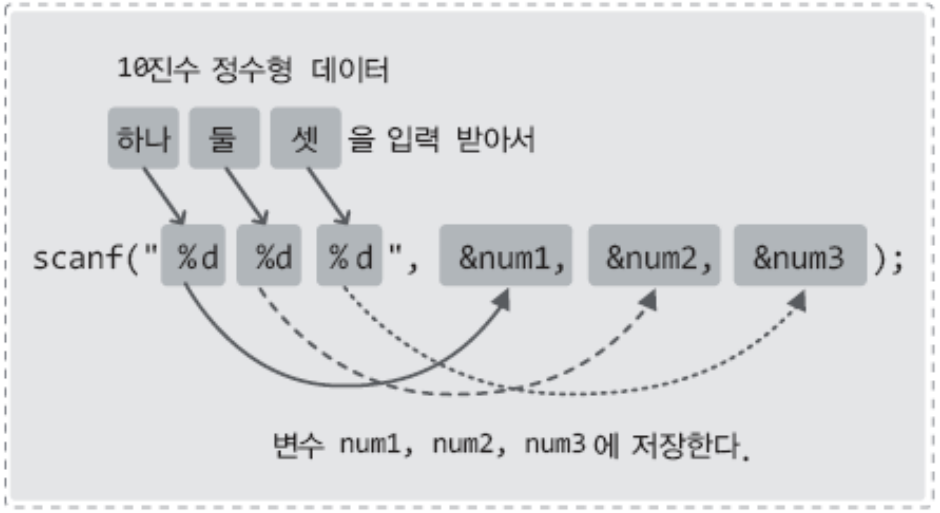
키보드로 정수 입력 : scanf 함수

```
#include <stdio.h>

int main(void)
{
    int result;
    int num1, num2;

    printf("정수 one: ");
    scanf("%d", &num1);          // 첫 번째 정수 입력
    printf("정수 two: ");
    scanf("%d", &num2);          // 두 번째 정수 입력

    result = num1 + num2;
    printf("%d + %d = %d \n", num1, num2, result);
}
```




```
정수 one: 3
정수 two: 4
3 + 4 = 7
```

입력의 형태를 다양하게 지정 가능




```
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);
}
```

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

C 언어의 표준 키워드

- 키워드
  - C 언어의 문법을 구성하는, 그 의미가 결정되어 있는 단어들!

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			

scanf 함수와 scanf_s 함수 (1)

- scanf() 함수
  - 표준 C의 함수
- scanf_s() 함수
  - 보안 상의 목적으로 Visual C++에서 만든 함수
  - 주로 문자나 문자열 입력 시 사용
    - 문자열 버퍼의 크기를 매개변수로 함께 전달함

strcpy, scrcpy_s 등 주로 문자열 처리 관련 함수들이 이와 같이 표준 외의 별도의 함수가 준비되어 있음

```
int main(void)
{
    char str[10];

    printf("문자열 입력 : ");
    scanf_s("%s", str, sizeof(str));    // scanf("%s", str);

    printf("입력 문자열 : %s\n", str);
}
```

문자열에 대해서는 추후 배우게 됨

scanf 함수와 scanf_s 함수 (2)

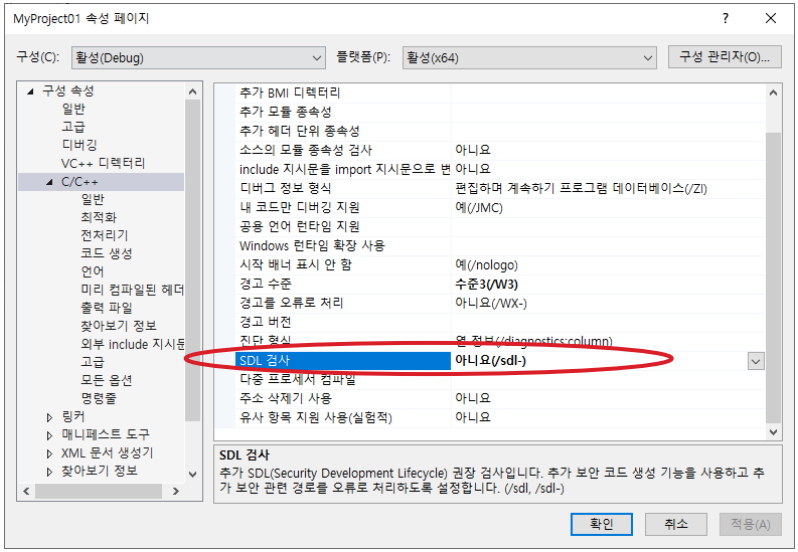
- Visual C++에서는 scanf 함수 사용 시 scanf_s 함수를 사용하도록 에러를 발생시킴
  - 에러 방지 방법 (다음 중 하나)

1. [프로젝트]-[...속성]-[C/C++]
   - SDL 검사 : 아니요
2. 소스코드 시작 부분에
   - 다음 코드 추가

```
#pragma warning(disable:4996)
```

3. 또는 시작 부분에
   - 다음 코드 추가

```
#define _CRT_SECURE_NO_WARNINGS
```



이번 장에서 배운 것

- 값(들)에 대해 연산이 요구될 때 연산자를 사용할 수 있다.
- 연산자에는 대입 연산자, 산술 연산자, 관계 연산자, 논리 연산자 등이 있다.
- 연산자들 사이에는 우선 순위가 있고 동일한 우선 순위를 가진 연산자들 사이에는 결합 방향이 정해져 있다.
- 값을 저장하기 위해서는 변수를 사용한다.
- 정수를 저장하기 위한 변수 타입과 실수를 저장하기 위한 변수 타입이 있다.
- scanf 함수를 사용하여 키보드를 통해 값을 입력받을 수 있다.
- Visual C++에서는 표준 C 함수인 scanf 함수를 사용하기 위해서는 프로젝트 생성 시 SDL을 체크하지 않으면 된다.