Chapter 2. Types, Operators, and Expressions

March, 2016

Seungjae Baek

Dept. of software
Dankook University
http://embedded.dankook.ac.kr/~baeksj

이 장의 강의 목표

- 변수의 이해
- C언어의 표준 키워드
- 연산자 소개
- 키보드 입력

변수의 이해 (1/9)

- 덧셈 예제

```c
#include <stdio.h>

int main(void)
{
    3 + 4;
    return 0;
}
```

Results

```
...
```

- ✓ 연산자 (Operator)
  - +, -, *, / …
- ✓ 변수의 (variable) 필요성

변수의 이해 (2/9)

- 변수 (Variable) ?
  - ✓ 값 또는 연산결과를 저장할 수 있는 메모리 공간의 이름
  - ✓ 변수를 선언하면 메모리 공간이 할당되고 이름이 붙음

- 변수의 이름
  - ✓ 이름을 통해서 메모리 공간에 접근이 가능
  - ✓ 메모리 공간에 값을 저장할 수 있고, 저장된 값을 참고할 수 있음

변수의 이해 (3/9)

- 변수 사용의 예

```c
#include <stdio.h>

int main(void)
{
    int num;
    num = 20;
    printf("%d\n", num);
    return 0;
}
```

```
int num;
-  int : 정수의 저장을 위한 메모리 공간의 할당
-  num : 할당된 메모리 공간의 이름
```

```
num = 20;
-  변수 num 공간에 20을 저장
```

```
printf("%d\n", num);
-  변수 num 공간의 20을 참조
```

변수의 이해 (5/9)

- 변수 선언 시 주의 사항

```c
#include <stdio.h>

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

```c
#include <stdio.h>

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

    int num2;
    num2 = 0;
    return 0;
}
```

- ✓ 과거 C 표준
  - 함수의 맨 처음에 변수의 초기화 필요
  - 1999년 C언어 표준에서는 위치 제한이 없어짐

변수의 이해 (6/9)

- 변수 이름의 규칙
  - ✓ 1. 알파벳, 숫자, 언더바(_)로 구성
  - ✓ 2. C 언어는 대소문자 구문 (ex: num 과 Num은 다른 변수)
  - ✓ 3. 변수의 이름은 숫자로 시작할 수 없고, 키워드를 변수 이름으로 사용할 수 없음
  - ✓ 4. 이름과 이름 사이에는 공백이 삽입될 수 없음

변수의 좋은 예

```
int num1;
int num2;
int Sum;
int CountOfBook;
```

변수의 나쁜 예

```
int 5Dogs;
int phone#;
int Count Of Book;
int int;
```

변수의 이해 (4/9)

- 변수의 다양한 선언 및 초기화 방법

```c
#include <stdio.h>
int main(void)
{
    int num1, num2;
    int num3=30, num4=40;
    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);
    return 0;
}
```

- ✓ int num1, num2;
  - 변수 선언 (값 할당을 안 함 – 이 경우 의미 없는 값 할당)
  - , 를 이용하여 둘 이상의 변수 선언 가능
- ✓ int num3=30, num4=40;
  - 선언과 동시에 초기화

변수의 이해 (7/9)

- 변수의 자료형 (Data Type)
  - ✓ 정수형 변수
    - 정수 값 저장을 목적으로 선언된 변수
    - char, short, int, long 형 변수로 구분
  - ✓ 실수 형 변수
    - 실수 값 저장을 목적으로 선언된 변수
    - float, double 형 변수로 구분

```
...
    short num1 = 30;      // 정수형 중 short 형 변수
    int num2 = 30;        // 정수형 중 int 형 변수
    long num3 = 30;       // 정수형 중 long 형 변수

    float data1 = 3.14;   // 실수형 중 float 형 변수
    double data2 = 3.14; // 실수형 중 double 형 변수
```

변수의 이해 (8/9)

- 덧셈 예제 – update 1

```c
#include <stdio.h>

int main(void)
{
    int num1 = 3;
    int num2 = 4;
    int result = num1 + num2;

    printf("Add result: %d \n", result);
    printf("%d + %d = %d \n", num1, num2, result);
    printf("Sum of %d and %d is %d.\n", num1, num2, result);
    return 0;
}
```

Results

```
Add result: 7
3 + 4 = 7
Sum of 3 and 4 is 7.
```

변수의 이해 (9/9)

- 연습 문제
  - ✓ 두 변수 `int x = 4, y = 2;` 이용하여 사칙 연산 결과를 출력하시오.
  - ✓ 사칙연산의 결과를 보관할 변수 4개 사용할 것

Results

```
Plus     : 4 + 2 = 6
Minus    : 4 - 2 = 2
Divide   : 4 / 2 = 2
Multiply : 4 x 2 = 8
```

Q: `int x = 5`로 바꾸면 연산 결과는 바뀌게 된다.
이 경우 문제는 없는가**?** 그 이유는 무엇인가**?**

C 언어의 표준 키워드(Keyword)

- 키워드 (keyword)
  - ✓ C언어의 문법을 구성하는 단어
  - ✓ 변수 또는 함수 이름으로 사용할 수 없음

auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volitile	while

연산자 소개 (1/15)

- 연산자 (Operator)
  - ✓ 컴파일러가 특정 계산 및 논리 연산을 할 수 있게 알려주는 기호

  - ✓ 산술 연산자 (Arithmetic Operators)
  - ✓ 관계 연산자 (Relational Operators)
  - ✓ 논리 연산자 (Logical Operators)
  - ✓ 비트 연산자 (Bitwise Operators)
  - ✓ 대입 연산자 (Assignment Operators)
  - ✓ 기타 연산자 (Misc Operators)

연산자 소개 (2/15)

- 대입 연산자와 산술 연산자
  (Assignment and Arithmetic Operators)
  - ✓ 이항 연산자

연산자	설명	예
=	연산자의 오른쪽 피연산자를 왼쪽 피연산자에게 할당	C = A + B A+B의 값을 C에게 할당
+	두 피연산자를 더한다	A+B의 값은 30
-	왼쪽 피연산자에서 오른쪽 피연산자를 뺀다	A-B의 값은 -10
*	두 연산자를 곱한다	A*B의 값은 200
/	왼쪽 피연산자를 오른쪽 피연산자로 나눈다	B/A의 값은 2
%	왼쪽 피연산자를 오른쪽 피연산자로 나누었을 때 나머지 값	B%A의 값은 0

  - ✓ A = 10, B = 20

연산자 소개 (3/15)

- 덧셈 예제 – update 1

```c
#include <stdio.h>

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("%quotient of %d ÷ %d =%d \n", num1, num2, num1/num2);
    printf("remainder of %d ÷ %d = %d \n", num1, num2, num1%num2);
    return 0;
}
```

Results

```
9 + 2 = 11
9 − 2 = 7
9 × 2 = 18
quotient of 9 ÷ 2 =4
remainder of 9 ÷ 2 = 1
```

연산자 소개 (4/15)

- 복합 대입 연산자 (Assignment Operators)

복합 연산자	일반 연산자 예	복합 연산자 예
+=	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

Q: 왜 복합 대입연산자를 사용할까?

```c
#include <stdio.h>
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;
}
```

Results

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

연산자 소개 (5/15)

- 부호 연산 의미를 갖는 +, -

```c
#include <stdio.h>
int main(void)
{
    int num1 = +2;
    int num2 = -4;
    num1 = -num1;
    printf("num1: %d \n", num1);
    num2 =- num2;
    printf("num2: %d \n", num2);
    return 0;
}
```

```c
num1 = -num1; // 부호
num1 -= num1; // 복합 대입
```

```c
num2 =- num2; // 부호
num2 -= num2; // 복합 대입
```

Results

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

연산자 소개 (6/15)

- 증가, 감소 연산자 (Arithmetic Operators)

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

연산자 소개 (7/15)

- 증가, 감소 연산자 (Arithmetic Operators)
  - ✓ 예제 1

```c
#include <stdio.h>
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);
    return 0;
}
```

Results

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

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

연산자 소개 (8/15)

- 증가, 감소 연산자 (Arithmetic Operators)
  - ✓ 예제 2

```c
#include <stdio.h>

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

    printf("num1: %d \n", num1);
    printf("num2: %d \n", num2);
    return 0;
}
```

Results

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

Q: 연산 과정을 단계별로 설명하시오.

연산자 소개 (9/15)

관계 연산자 (Relational Operators)

연산자	설명	예
==	두 피연산자의 값이 같은지 비교 같으면 true, 다르면 false	(A==B) is not true
!=	두 피연산자의 값이 다른지 비교 다르면 true, 같으면 false	(A!=B) is ture
>	왼쪽 피연자가 오른쪽 보다 큰지 확인 크면 true, 작으면 false	(A>B) is not ture
<	왼쪽 피연자가 왼쪽 보다 작은지 확인 작으면 true, 크면 false	(A<B) is ture
>=	왼쪽 피연자가 오른쪽 보다 크거나 같은지 확인 크거나 같으면 true, 작으면 false	(A>=B) is not ture
<=	왼쪽 피연자가 오른쪽 보다 작거나 같은지 확인 작거나 같으면 true, 크면 false	(A<=B) is ture

- ✓ 예) A=3; B=4; (A==B)?

연산자 소개 (10/15)

- 관계 연산자 (Relational Operators)
  - ✓ 예제

```c
#include <stdio.h>

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;
}
```

Results

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

연산자 소개 (11/15)

- 논리 연산자 (Logical Operators)

연산자	설명	예
&&	논리 AND 피 연산자 두 개 모두 0이 아닐 때 true	(A&&B) is false
||	논리 OR 피 연산자 둘 중에 하나만 0이 아니면 ture	(A||B) is ture
!	논리 NOT 피 연산자의 논리 상태를 반대로 변경	!A is false !(A&&B) is ?

  - ✓ 예) A = 1, B = 0, (A&&B)?

연산자 소개 (12/15)

- 논리 연산자 (Logical Operators)
  - ✓ 예제

```c
#include <stdio.h>

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);
    return 0;
}
```

Results

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

연산자 소개 (13/15)

- 콤마 연산자 (,)
  - ✓ 둘 이상의 변수를 동시 선언
  - ✓ 둘 이상의 문장을 한 행에 삽입하는 경우
  - ✓ 둘 이상의 인자를 함수로 전달할 때도 인자

```c
#include <stdio.h>

int main(void)
{
    int num1=1, num2=2;
    printf("Hello "), printf("world! \n");
    num1++, num2++;
    printf("%d ", num1), printf("%d ", num2), printf("\n");
    return 0;
}
```

Results

```
Hello world!
2 3
```

연산자 소개 (14/15)

- 연산자의 우선 순위와 결합 방향
  - ✓ 우선 순위
    - 연산의 순서에 대한 순위
    - 덧셈과 뺄쎔 보다는 곱셈과 나눗셈의 우선 순위가 높음
  - ✓ 결합 방향
    - 우선순위가 동일한 두 연산자 사이에서의 연산을 진행하는 방향
    - 덧셈, 뺄셈, 곱셈, 나눗셈 모두 결합방향이 왼쪽에서 오른쪽으로 진행

연산자 예

```
1. -3 + 4 * 5 - 6

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

Q: 어떤 순서로 계산 해야 하는가 **?**

연산자 소개 (15/15)

연산자의 우선 순위

연산자	설명	결합 방향
( ) [ ] . -> ++ --	함수 호출 (Parentheses) 인덱스 (Brackets – array subscript) 직접 지정 (Member selection via object name) 간전 지정 (Member selection via pointer) 후위 증가, 감소 (Postfix increment/decrement)	left-to-right
++ -- + - ! ~ (type) * & sizeof	전위 증가, 감소 (Prefix increment/decrement) 부호 연산 – 음수, 양수 (Unary plus/minus) 논리 NOT / 비트 단위 NOT (Logical negation/bitwise complement) 자료형 변환 (Cast) (convert value to temporary value of type) 간전지정 연산 (Dereference) 주소 연산 (Address) (of operand) 바이트 단위 크기 계산 (Determine size in bytes on this implementation)	right-to-left
* / %	산술 연산 (Multiplication/division/modulus)	left-to-right
+ -	산술 연산 (Addition/subtraction)	left-to-right
<< >>	비트 이동 (Bitwise shift left, Bitwise shift right)	left-to-right
< <= > >=	관계 연산자 (Relational less than/less than or equal to) 관계 연산자 (Relational greater than/greater than or equal to)	left-to-right
== !=	관계 연산자 (Relational is equal to/is not equal to)	left-to-right
&	비트 연산 (Bitwise AND)	left-to-right
^	비트 연산 (Bitwise exclusive OR)	left-to-right
|	비트 연산 (Bitwise inclusive OR)	left-to-right
&&	논리 연산 (Logical AND)	left-to-right
||	논리 연산 (Logical OR)	left-to-right
? :	조건 연산(Ternary conditional)	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	대입 연산 (Assignment) Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	right-to-left
,	Comma (separate expressions)	left-to-right

키보드 입력 (1/5)

- SimpleAddTwo 프로그램 (PPT. 10)

```c
#include <stdio.h>

int main(void)
{
    int num1 = 3;
    int num2 = 4;
    int result = num1 + num2;

    printf("Add result: %d \n", result);
    printf("%d + %d = %d \n", num1, num2, result);
    printf("Sum of %d and %d is %d.\n", num1, num2, result);
    return 0;
}
```

Q: 이 프로그램의 문제점은 무엇일까요 **?**

키보드 입력 (2/5)

- SimpleAddTwo 프로그램 업그레이드
  - ✓ scanf 함수 사용 (1)
    - 하나의 값 입력 받기

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

scanf 사용법

```
scanf("%d", &num);
```

2. 입력한 값을 **num** 변수에 저장

1. 10진수로 사용자가 입력

Q: printf 와 **scanf** 사용법이 어떻게 다른가요**?**

키보드 입력 (3/5)

- SimpleAddTwo 프로그램 업그레이드
  - ✓ 2개의 정수 입력 후 합을 구하는 예

```c
#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);
    return 0;
}
```

Results

```
정수 one: 1
정수 two: 2
1 + 2 = 3
```

키보드 입력 (4/5)

- SimpleAddTwo 프로그램 업그레이드
  - ✓ 3개의 정수 입력 후 합을 구하는 예

Q: 값을 입력할 때 공백을 많이 주거나 enter를 입력하면?

```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;
}
```

Results

```
세 개의 정수 입력: 1 2 3
1 + 2 + 3 = 6
```

키보드 입력 (5/5)

- 연습문제
  - ✓ scanf 함수를 사용하여 정수를 입력 받고 해당하는 구구단을 출력
  - ✓ 입력 받은 정수는 (1~9)

Results

```
원하는 구구단 (1~9): 5
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
```

이 장의 결론

- 변수의 이해
  - ✓ 변수의 사용 이유, 메모리의 구조
- C언어의 표준 키워드
- 연산자 소개
  - ✓ 연산자 우선 순위
- 키보드 입력
  - ✓ scanf 함수