변수와 연산자

- 변수, 연산자, 데이터 입력 -

성공회대학교 IT융합자율학부
소프트웨어공학전공
홍 성 준

연산을 위한 연산자와 값의 저장을 위한 변수

◎ 덧셈 프로그램의 구현에 필요한 '+' 연산자

- SimpleAddOne.c

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

  - 아무 문제 없이 컴파일에 성공.
  - 그러나 실행 결과는?

◎ 연산자 (operator)

- C언어를 이용해서 특정 연산을 하고자 할 때 사용하는 약속된 기호

"덧셈 연산을 하고 그 결과를 메모리 공간에 저장한다. 그리고 메모리 공간에 저장된 값을 출력한다."

연산을 위한 연산자와 값의 저장을 위한 변수

◎ 변수를 이용한 데이터의 저장

- 변수(variable)
  - 값을 저장할 수 있는 메모리 공간에 붙여진 이름
  - 변수를 선언하면 변수 이름을 통해 메모리에 값을 저장하거나 저장된 값을 참조하거나 변경할 수 있음

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

int num

- int　　정수의 저장을 위한 메모리 공간의 할당
- num　　할당된 메모리 공간의 이름은 num

num=20;

- 변수 num에 접근하여 20을 저장

printf("%d", num);

- num에 저장된 값을 참조(출력)

◎ 대입 연산자 '='

- 대입 연산자 오른편에 있는 값을 왼편의 변수에 저장하는 대입 연산

연산을 위한 연산자와 값의 저장을 위한 변수

◎ 변수의 다양한 선언 및 초기화 방법

- 초기화 : 변수를 선언하고 처음으로 값을 저장하는 것
- 대입 연산 : 초기화 이후 저장된 값을 변경하여 저장하는 것

◎ VarDeclAndInit.c

```
int main(void)
{
    int num1, num2;     // 변수 num1, num2의 선언
    int num3=30, num4=40;    // 변수 num3, num4의 선언 및 초기화

    printf("num1: %d, num2: %d \n", num1, num2);
    num1=10;    // 변수 num1의 초기화
    num2=20;    // 변수 num2의 초기화

    printf("num1: %d, num2: %d \n", num1, num2);
    printf("num3: %d, num4: %d \n", num3, num4);
    return 0;
}
```

int num1, num2;

- 변수를 선언만 할 수 있다.
- 콤마를 이용하여 둘 이상의 변수를 동시에 선언할 수 있다.
- 선언만 하면 값이 대입되기 전까지 쓰레기 값(의미 없는 값)이 채워진다.

int num3=30, num4=40;

- 선언과 동시에 초기화 할 수 있다.

실행결과

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

연산을 위한 연산자와 값의 저장을 위한 변수

◎ 변수 선언 시 주의할 점

- 중괄호 내에 변수를 선언할 경우, 변수의 선언문은 중괄호의 앞부분에 위치해야 함

```
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;
    . . . .
}
```

컴파일이 불가능할 수도 있는 변수선언

◎ 변수의 명명 규칙

- 변수 이름은 알파벳, 숫자, '_' (언더바)로만 구성 가능함
- C언어는 대소문자를 구문하기 때문에, num과 Num은 다른 변수로 인식
- 변수 이름은 숫자로 시작할 수 없고, 키워드도 변수의 이름으로 사용할 수 없음
- 변수 이름 간에는 띄어쓰기를 포함할 수 없음

```
int 7ThVal;
int phone#;
int your name;
```

연산을 위한 연산자와 값의 저장을 위한 변수

◎ 변수의 자료형 (data type)

- 정수형 변수
  - 정수값 저장을 목적으로 선언된 변수

  (예) char 형, short 형, int 형, long 형 등
- 실수형 변수
  - 실수값 저장을 목적으로 선언된 변수

  (예) float 형, double 형 등

int num1=24

· num1은 정수형 변수 중 int형 변수

double num2=3.14

· num2는 실수형 변수 중 double형 변수

연산을 위한 연산자와 값의 저장을 위한 변수

◎ 덧셈 프로그램의 완성

- SimpleAddTwo.c

```
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);
    return 0;
}
```

C언어의 다양한 연산자 소개

◎ 대입 연산자와 산술 연산자

- 두 개의 피연산자를 갖는 이항 연산자 (binary operator)

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;	→

C언어의 다양한 연산자 소개

◎ 대입 연산자와 산술 연산자 (cont.)

- OperatorOne.c

```
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);
    return 0;
}
```

- "함수 호출문의 인자전달 위치에 연산식이 올 수 있다."

C언어의 다양한 연산자 소개

◎ 복합 대입 연산자

- OperatorTwo.c

```
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;
}
```

C언어의 다양한 연산자 소개

◎ 부호 연산의 의미를 갖는 '+' 연산자와 '-' 연산자

- 숫자나 변수의 부호를 의미하는 단항 연산자 (unary operator)
- OperatorThree.c

```
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;
}
```

```
num1=-num2;    // 부호 연산자의 사용
num1-=num2;    // 복합 대입 연산자의 사용
```

```
num1 = -num2;    // 부호 연산자의 사용
num1 -= num2;    // 복합 대입 연산자의 사용
```

C언어의 다양한 연산자 소개

◎ 증가, 감소 연산자

- OperatorFour.c

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

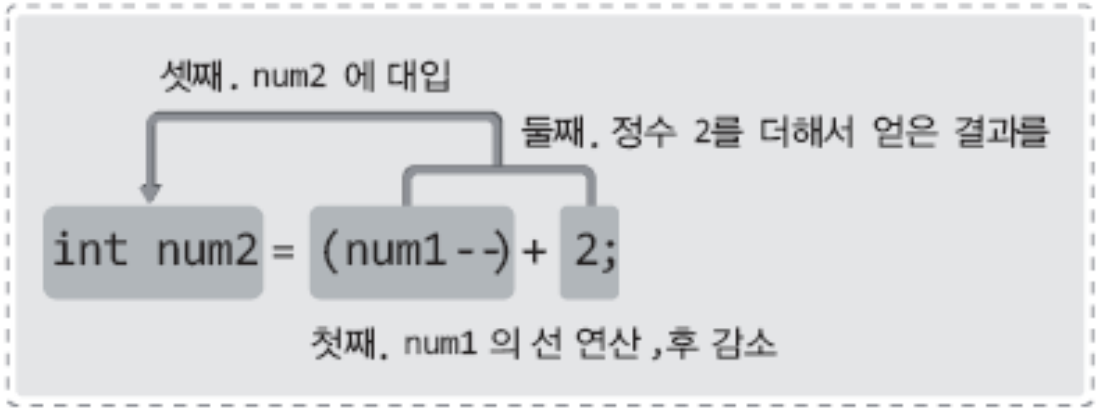
연산자	연산자의 기능	결합방향
++num	값을 1 증가 후, 속한 문장의 나머지를 진행(선 증가, 후 연산) 예) val = ++num;	←
num++	속한 문장을 먼저 진행한 후, 값을 1 증가(선 연산, 후 증가) 예) val = num++;	←
--num	값을 1 감소 후, 속한 문장의 나머지를 진행(선 감소, 후 연산) 예) val = --num;	←
num--	속한 문장을 먼저 진행한 후, 값을 1 감소(선 연산, 후 감소) 예) val = num--;	←

C언어의 다양한 연산자 소개

◎ 증가, 감소 연산자 (cont.)

- OperatorFive.c

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

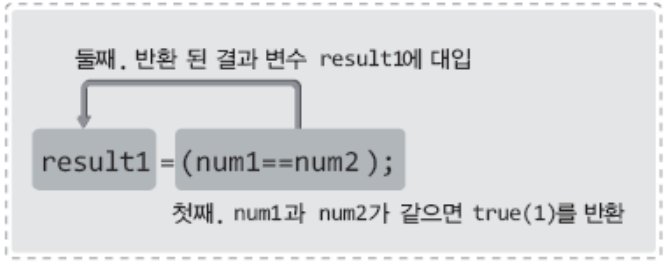


C언어의 다양한 연산자 소개

◎ 관계 연산자 (비교 연산자)

- 대소와 동등의 조건을 따지는 연산자
- 연산의 조건을 만족하면 참(1)을 반환하고 만족하지 않으면 거짓(0)을 반환하는 연산자
- OperatorSix.c

```
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;
}
```



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?	→

C언어의 다양한 연산자 소개

◎ 논리 연산자

- AND(논리곱), OR(논리합), NOT(논리부정)을 표현하는 연산자

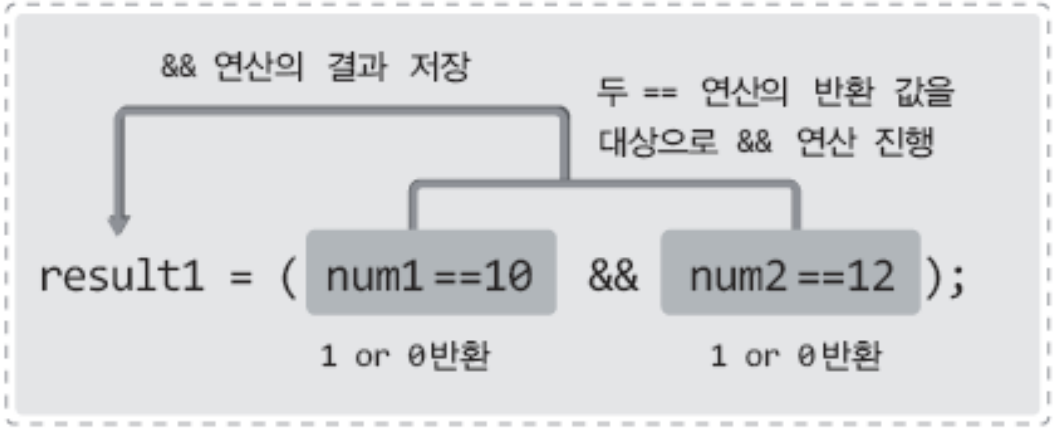
연산자	연산자의 기능	결합방향
&&	예) A && B A와 B 모두 '참'이면 연산결과로 '참'을 반환(논리 AND)	→
||	예) A || B A와 B 둘 중 하나라도 '참'이면 연산결과로 '참'을 반환(논리 OR)	→
!	예) !A A가 '참'이면 '거짓', A가 '거짓'이면 '참'을 반환(논리 NOT)	←

C언어의 다양한 연산자 소개

◎ 논리 연산자 (cont.)

- OperatorSeven.c

```
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;
}
```



C언어의 다양한 연산자 소개

◎ 콤마 연산자 ',‘

- 두 개 이상의 변수를 동시에 선언할 때 사용
- 두 개 이상의 명령 구문을 한 줄에 삽입할 때 사용
- 두 개 이상의 인자를 함수로 전달할 때 사용
- 다른 연산자와는 다르게 특정 연산이 아닌 ‘구분’을 목적으로 함
- CommaOp.c

```
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;
}
```

C언어의 다양한 연산자 소개

◎ 연산자의 우선순위

- 연산의 순서에 대한 순위

◎ 연산자의 결합 방향

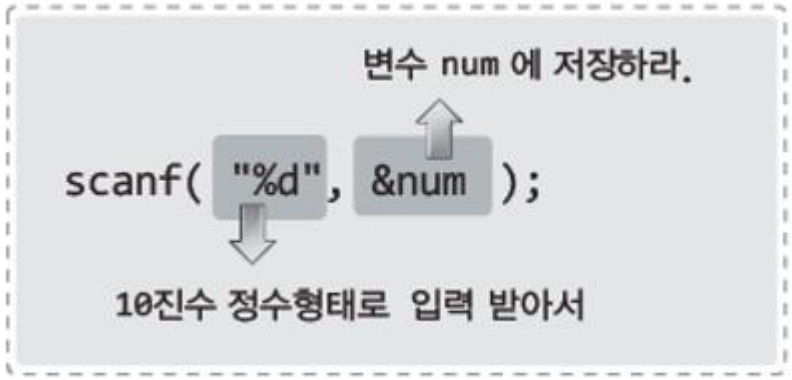
- 우선순위가 동일한 두 연산자가 하나의 수식에 존재하는 경우 순서를 결정해 놓은 것

우선순위	연산자	설명	순위가 같을 경우 진행방향
1	( ) [ ] . ->	1차 연산자	→
2	+ - ++ -- ~ ! * &	단항 연산자, 변수(또는 상수) 앞에 붙음	←
3	* / %	산술 연산자	→
4	+ -	산술 연산자	→
5	<< >>	비트 시프트 연산자	→
6	<<= >>=	비교 연산자	→
7	== !=	동등 연산자	→
8	&	비트 연산자	→
9	^	비트 연산자	→
10	|	비트 연산자	→
11	&&	논리 연산자	→
12	||	논리 연산자	→
13	?:	삼항 연산자	→
14	= += -= *= /= %= &= ^= |= <<= >>=	대입 연산자	←
15	,	콤마 연산자	→

키보드로부터의 정수입력을 위한 scanf 함수의 호출

◎ scanf() 함수의 기본적인 사용 방법

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



키보드로부터의 정수입력을 위한 scanf 함수의 호출

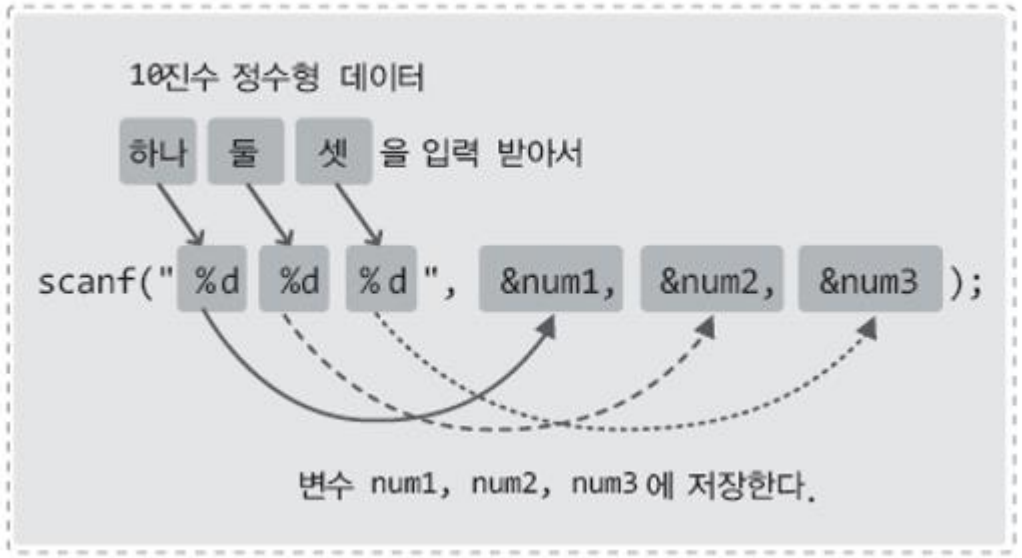
◎ SimpleAddThree.c

```
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;
}
```

키보드로부터의 정수입력을 위한 scanf 함수의 호출

◎ scanf() 함수의 다양한 입력 형태

- 서식 문자를 이용하여 여러 개의 데이터를 다양한 형태로 입력 받을 수 있음



키보드로부터의 정수입력을 위한 scanf 함수의 호출

◎ SimpleAddFour.c

```
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;
}
```

C언어의 표준 키워드 (keyword)

◎ 키워드

- 이미 기능적 의미가 정해져 C언어의 문법을 구성하는 단어
- 프로그래머가 다른 용도로 사용할 수 없음 (예) 변수명, 함수명, 상수명 등

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			