일일 업무 사항 정리

작성자	제품팀 이민성 인턴
업무 일시	20220919 ~ 20220923

세부 사항

1. 업무 내역 요약 정리

목표 내역	Done & Plan
1 주차) C 언어 개요 2 주차) 연산자/제어문 3 주차) 함수 4 주차) 배열/포인터 5 주차) 구조체/공용체 6 주차) 전처리기	1. 연산을 위한 연산자와 값의 저장을 위한 변수 - 덧셈 프로그램의 구현에 필요한 + 연산자 - 변수를 이용한 데이터의 저장 - 변수의 다양한 선언 및 초기화 방법 - 변수선언 시 주의할 사항 - 변수의 자료형 - 덧셈 프로그램의 완성 2. C언어의 다양한 연산자 소개 - 대입 연산자와 산술 연산자 - 복합 대입 연산자 - 부호연산의 의미를 갖는 + 연산자와 - 연산자 - 증가, 감소 연산자 - 관계 연산자 - 논리 연산자 - 콤마 연산자 - 연산자의 우선순위와 결합방향 3. 키보드로부터의 데이터 입력과 C언어의 키워드 - 키보드로부터의 정수입력을 위한 scanf 함수의 호출 - 입력의 형태를 다양하게 지정할 수 있다 4. 문제 5. 제어문

2. 내용 세부 (업무 세부 내역 정리 및 기타 사항 정리)

연산자 / 제어문

1. 연산을 위한 연산자와 값의 저장을 위한 변수

- 덧셈 프로그램의 구현에 필요한 + 연산자

```c
#include <stdio.h>

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

위 예제는 문제없이 컴파일 된다. 이는 C언어가 + 기호를 지원한다는 뜻이 된다. 지원하지 않으면 컴파일 에러가 발생하게 된다. 이렇듯 C언어를 이용해서 특정연산을 요구할 때 사용하는 약속된 기호를 가리켜 '**연산자(operator)**'라 한다.

위 예제를 실행한다면 덧셈의 결과가 어디에도 출력되지 않음을 알 수 있다. 하지만 이는 당연한 것이다. 프로그램 내에 덧셈을 요구하는 문장은 있지만, 출력을 요구하는 문장은 존재하지 않기 때문이다. 말 그대로 덧셈만 시켰을 뿐, 그 이후에 대해서는 아무런 언급도 되어있지 않다. 그래서 다음의 형태로 프로그램을 변경하고자 한다.

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

위에서 언급한 것처럼 덧셈결과를 저장하게 되면, printf 함수의 호출을 통해서 다양한 형태로 출력하는 것이 가능해질 뿐만 아니라, 저장된 값을 가지고 추가적인 연산도 진행할 수가 있다. 그렇다면 덧셈결과는 어떻게 저장해야 할까?

- 변수를 이용한 데이터의 저장

변수: 값을 저장할 수 있는 메모리 공간에 붙은 이름, 혹은 메모리 공간 자체를 가리키는 것

따라서 변수라는 것을 하나 만들면(선언하면), 그 변수의 이름을 통해서 값의 저장 및 참조가 가능하고, 또 저장된 값의 변경도 가능하다. 그럼 변수의 선언방법을 보이겠다.

```c
#include <stdio.h>

int main(void)
{
   int num; // num이라는 이름의 변수선언

}
```

위의 코드에서 보이듯이 int num; 문장이 변수의 선언문이다. 이 문장을 구성하는 int와 num이 의미하는 바는 각각 다음과 같다.

int: 정수의 저장이 가능한 메모리 공간을 할당한다.

num: 그리고 그 메모리 공간의 이름을 num이라 한다.

따라서 아래의 코드에서 보이듯이 변수 num을 선언한 다음에는 num이라는 이름을 이용해서 값을 저장하고 참조할 수 있다.

```c
#include <stdio.h>

int main(void)
{
   int num; // num이라는 이름의 변수선언
   num = 20; // 변수 num에 20 저장
   printf("%d", num); // 변수 num의 값 참조
}
```

C언어에서 = 기호는 값의 대입을 뜻한다. 이 기호를 가리켜 '대입 연산자'라 하며, 대입 연산자의 오른편에 오는 값을 왼편에 오는 변수에 저장하는 형태로 사용이 된다. 따라서 위의 코드를 실행하면 num에 저장된 값 20 이 출력되는 것을 확인할 수 있다.

- 변수의 다양한 선언 및 초기화 방법

선언된 변수에 처음 값을 저장하는 것을 가리켜 '초기화'라 한다. 그리고 초기화 이후에 저장된 값을 변경할 때에는 그냥 '대입' 또는 '대입연산'을 진행한다고 한다.

```
#include <stdio.h>

int main(void)
{
   int num; // num이라는 변수의 선언
   num = 12; // 변수 num을 12로 초기화
   num = 24; // 변수 num에 24를 대입
}
```

그런데 C언어에서는 다음과 같이 변수를 **'선언과 동시에 초기화'** 하는 것이 가능하다.

```
int num = 12;
```

위의 문장을 실행하게 되면, 변수 num이 메모리 공간에 할당되자마자 12 로 초기화된다. 그리고 다음과 같이 둘 이상의 변수를 동시에 선언하는 것도 가능하고, 동시에 선언 및 초기화하는 것도 가능하다.

```
int num1, num2; // 두 개의 변수를 선언
int num3 = 20, num4 = 40; // 두 개의 변수를 선언 및 초기화
```

- VarDeclAndInit.c

```
#include <stdio.h>

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;
}
```

```
[lms@localhost c]$ ./ VarDeclAndInit
num1 : 32764, num2 : 2139121312
num1 : 10, num2 : 20
num3 : 30, num4 : 40
```

위의 VarDeclAndInit.c는 변수의 선언 및 초기화 방법뿐만 아니라 다음의 사실도 말해준다.

"변수를 선언만 하고 초기화하지 않으면 쓰레기 값이 저장된다."

여기서 말하는 쓰레기 값이란 **'아무런 의미가 없는 값'**을 뜻한다. 그리고 가급적 이러한 쓰레기 값이 변수에 저장되지 않도록 하는 것이 좋다. 따라서 다음과 같이 변수를 선언과 동시에 0 으로 초기화한 다음, 이후에 의미 있는 값을 저장하기도 한다.

```
int num1 = 0, num2 = 0;
```

하지만 이는 절대적으로 지켜야 할 규칙은 아니다.

- 변수선언 시 주의할 사항

첫번째:

"중괄호 내에 변수를 선언할 경우, 변수의 선언문은 중괄호의 앞부분에 위치해야 한다."

다음과 같은 코드는 컴파일 에러를 발생시키지 않는다.

```
#include <stdio.h>

int main(void)
{
   int num1;
   int num2;
   num1=0; // 변수선언 이후에 등장한 초기화 문장
   num2=0; // 변수선언 이후에 등장한 초기화 문장
}
```

변수 num1 과 num2 의 선언문 앞에 다른 어떠한 문장도 존재하지 않기 때문에 컴파일 에러를 발생시키지 않는다. 하지만 다음과 같은 코드는 컴파일 에러를 발생시킨다.

```
#include <stdio.h>

int main(void)
{
   int num1;
   num1=0; // 이 문장은 변수의 선언문이 아니다.
   int num2; // 컴파일 에러가 발생하는 시점
   num2=0;
}
```

위의 경우 변수 num2 의 선언문 앞에 변수의 선언이 아닌 문장이 등장했기 때문에 컴파일 에러가 발생한다. 따라서 중괄호의 앞부분에 변수의 선언문이 오도록 해야한다.

두번째: 변수의 이름을 지을 때 적용되는 규칙

1. 변수의 이름은 알파벳, 숫자, 언더바(_)로 구성된다.
2. C언어는 대소문자를 구분한다. 따라서 변수 Num과 변수 num은 서로 다른 변수이다.
3. 변수의 이름은 숫자로 시작할 수 없고, 키워드도 변수의 이름으로 사용할 수 없다.
4. 이름 사이에 공백이 삽입될 수 없다.

```
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 이름을 정할 때에는 변수의 역할에 어울리는, 의미 있는 이름을 지어야한다."

- 변수의 자료형 (Date Type)

- 정수형 변수: 정수의 저장을 목적으로 선언된 변수
- 실수형 변수: 소수점 이하의 값을 지니는 실수의 저장을 목적으로 선언된 변수

변수의 종류가 크게 두 가지로 나뉘는 이유는 **정수냐, 실수냐에 따라서 값이 메모리 공간에 저장 및 참조 되는 방식이 다르기 때문**이다. 그리고 정수형 변수는 변수의 크기에 따라서 char형, short형, int형, long형 변수로 나뉘고, 실수형 변수도 크기에 따라서 float형 변수와 double형 변수로 나뉜다.

- 덧셈 프로그램의 완성

```c
#include <stdio.h>

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

    printf("덧 셈 결 과 : %d \n", result);
    printf("%d+%d=%d \n", num1, num2, result);
    printf("%d와  %d의 합 은  %d입 니 다 . \n", num1, num2, result);
    return 0;
}
```

```
덧 셈 결 과 : 7
3+4=7
3와  4의 합 은  7입 니 다 .
```

2. C언어의 다양한 연산자 소개

- 대입 연산자(=)와 산술 연산자(+,-,*,/,%)

두 개의 피연산자를 요구하는 연산자를 가리켜 '이항 연산자'라 하는데, 대입 연산자와 산술 연산자는 모두 이항 연산자들이다.

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;	→

```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("%d/%d의  몫 =%d \n",num1, num2, num1/num2);
    printf("%d/%d의  나 머 지 =%d \n",num1, num2, num1%num2);
    return 0;
```

```
9+2=11
9-2=7
9*2=18
9/2의  몫 =4
9/2의  나 머 지 =1
```

위 예제 6~10 행을 통해서 알 수 있는 사실은 다음과 같다.

"함수 호출문의 인자전달 위치에 연산식이 올 수 있다."

이러한 경우 함수의 호출에 앞서 연산식이 먼저 진행되며, 그 연산의 결과가 인자가 되어 함수의 호출까지 이어지는 것이다.

- 복합 대입 연산자

다른 연산자와 합쳐진 형태의 대입 연산자도 존재하는데, 이를 가리켜 복합 대입 연산자라 하며, 이의 종류는 다음과 같다.

***=, /=, %=, +=, -=, <<=, >>=, &=, ^=, |=**

이 중에서 산술 연산자와 합쳐진 복합 대입 연산자의 의미는 다음과 같다.

a = a + b	<----------동일 연산---------->	a += b
a = a – b	<----------동일 연산---------->	a -= b
a = a * b	<----------동일 연산---------->	a *= b
a = a / b	<----------동일 연산---------->	a /= b
a = a % b	<----------동일 연산---------->	a %= b

위 표에서 보듯이, a와 b의 덧셈결과를 다시 a에 저장하는 연산식 a=a+b는 a+=b와 같이 간략하게 표현하는 것이 가능하다. 따라서 += 연산자는 + 연산자와 = 연산자를 결합한 형태라고 말할 수 있다.

```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;
}
```

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

- 부호연산의 의미를 갖는 + 연산자와 – 연산자

+ 연산자와 – 연산자는 이항 연산자로서 덧셈과 뺄셈을 의미하지만, 피연산자가 하나인 단항 연산자로서 부호를 뜻하기도 한다. 이는 +3 와 -7 과 같이 숫자 앞에 붙는 부호를 뜻하는 것이다.

```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;
}
```

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

참고로, 부호 연산자가 삽입된 문장과 복합 대입 연산자가 삽입된 문장을 혼동하는 경우가 있다. 다음과 같이 연산자와 피연산자 사이에 공백을 삽입하지 않는 경우에는 더욱 혼동하기가 쉽다.

num1 = -num2; // 부호 연산자의 사용

num1 -= num2; // 복합 대입 연산자의 사용

따라서 모든 경우에 연산자와 피연산자 사이에 공백을 두지는 않더라도, 위의 두 경우에 한해서는 다음과 같이 공백을 두는 것이 혼란을 최소화할 수 있는 방법이 된다.

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

- 증가, 감소 연산자

이번에 소개하는 연산자는 변수에 저장된 값을 1 증가 및 감소시키는 경우에 사용되는 연산자이다. 단항 연산자로서 활용의 빈도가 높다.

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--;	→

```c
#include <stdio.h>

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;
}
```

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

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

```c
#include <stdio.h>

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

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

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

위 예제 6 행에서는 변수 num1 에 대해서 선 연산, 후 감소를 진행하고 있다. 그런데 이 연산의 앞과 뒤에 소괄호가 채워져 있다. C언어에서는 소괄호도 연산자이다. 그리고 이는 수학의 소괄호와 의미가 같다. 즉, 먼저 연산하라는 뜻이다. 따라서 다음의 순서대로 식이 진행된다.

첫째, num1 의 선 연산, 후 감소
둘째, 정수 2 를 더해서 얻은 결과를
셋째, num2 에 대입

그렇다면, 결과적으로 num2 에는 얼마가 저장될까? 아무리 선 연산, 후 감소라 할지라도, 괄호까지 해줬으니까 num1 의 값이 감소한 상태에서 덧셈연산이 진행되는 게 아닐까? 하지만 출력결과에서는 정수 2 와의 덧셈연산이 진행된 이후에 값이 감소했음을 보이고 있다. 즉, 후위 증가 및 후위 감소 연산 시에는 **소괄호의 영향을 받지 않고, 다음 문장으로 넘어가야만 비로소 값의 증가 및 감소가 이뤄진다** 라는 사실을 위 예제에서 보여주고 있다.

- 관계 연산자(<, >, ==, !=, <=, >=)

관계 연산자는 대소와 동등의 관계를 따지는 연산자이다. 예를 들어서 a와 b라는 숫자 또는 변수가 있다면, 둘이 같은지 다른지, 누가 더 크고 작은지를 따지는 연산자이다. 그래서 관계 연산자를 두고 '비교 연산자'라고도 한다. 두 개의 값을 비교하기 때문이다.

연산자	연산자의 기능	결합방향
<	예) n1 < n2 n1 이 n2 보다 작은가?	→
>	예) n1 > n2 n1 이 n2 보다 큰가?	→
==	예) n1 == n2 n1 과 n2 가 같은가?	→
!=	예) n1 != n2 n1 이 n2 가 다른가?	→
<=	예) n1 <= n2 n1 이 n2 보다 같거나 작은가?	→
>=	예) n1 >= n2 n1 이 n2 보다 같거나 큰가?	→

위의 관계 연산자들은 조건을 만족하면 1 을, 만족하지 않으면 0 을 반환한다. 그런데 여기서 말하는 1 은 참, 0 은 거짓을 의미하는 대표적인 숫자이다. 따라서 다음과 같이 이야기하는 것이 더 일반적이다.

"조건을 만족하면 참을, 만족하지 않으면 거짓을 반환한다."

```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;
}
```

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

9 행이 실행되는 과정:

첫째, num1 과 num2 가 같으면 true(1)를 반환

둘째, 반환 된 결과 변수 result1 대입

- 논리 연산자(&&, ||, !)

논리 연산자란 AND(논리곱), OR(논리합), NOT(논리부정)을 표현하는 연산자로서, 사용방법과 연산의 결과는 아래의 표에서 언급하는 바와 같다.

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

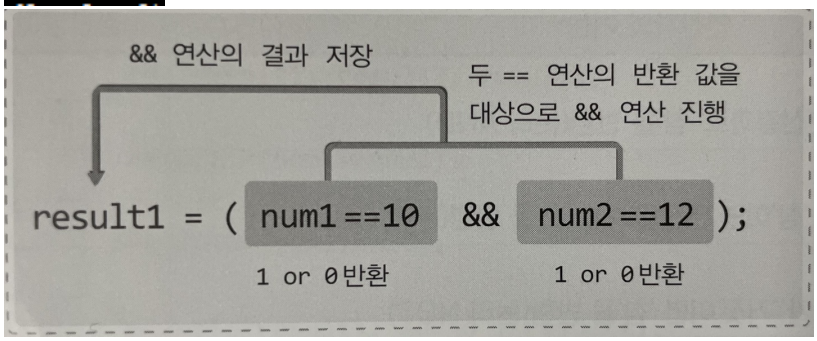
```c
#include <stdio.h>

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;
}
```

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



위 그림에서 설명하듯이 9 행에서는 관계연산의 결과를 대상으로 논리연산을 진행하고 있다. 즉, 1 차로 진행되는 것은 논리 연산자의 왼쪽과 오른쪽에 있는 관계 연산이다. 그리고 관계연산의 결과를 대상으로 논리연산을 진행하게 된다. 따라서 논리 연산자보다 관계 연산자가 먼저 진행이 된다는 것을 알 수 있다.

이렇듯 하나의 문장 안에 둘 이상의 연산자가 존재하는 경우 어떠한 연산을 먼저 진행하느냐에 대한 문제에 봉착하게 되는데, 이러한 부분의 혼란을 없애기 위해서 C언어는 연산자 우선순위라는 것을 정의하고 있다. 즉, 관계 연산자가 먼저 진행된 것도 연산자 우선순위에 근거한 결과이다.

result2 = (num1<12 || num2>12)

이 경우에도 <와 >연산의 결과를 대상으로 || 연산이 진행된다. 즉, 9 행과 연산의 순서가 동일하다. 마지막으로 11 행에 해당하는 다음 문장을 보자.

result3 = (!num1);

이 문장은 논리 NOT연산의 예를 보여준다. 따라서 num1 에 1 이 저장되어 있다면 result3 에는 0 이 저장되고,

num1 에 0 이 저장되어 있다면 result3 에는 1 이 저장된다. 그렇다면 0 도 1 도 아닌 다른 값이 저장되어 있다면 result3 에는 어떤 값이 저장되겠는가?

"C언어는 0 이 아닌 모든 값을 참으로 간주한다."

즉, 거짓을 의미하는 숫자는 유일하게 0 하나이고, 0 이 아닌 모든 숫자는 참으로 인식된다. 다만, 그 중에서도 참을 의미할 때 주로 사용되는 숫자가 1 이고, 그래서 연산의 결과로 참이 반환되어야 할 때 1 이 반환되는 것뿐이다.

- 콤마 연산자(,)

콤마 연산자는 둘 이상의 변수를 동시에 선언하거나, 둘 이상의 문장을 한 행에 삽입하는 경우에 사용되는 연산자이다. 뿐만 아니라, 둘 이상의 인자를 함수로 전달할 때도 인자의 구분을 목적으로 사용된다. 즉, 콤마 연산자는 다른 연산자들과 달리, 연산의 결과가 아닌 구분을 목적으로 주로 사용된다.

```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;
}
```

```
Hello world!
2 3
```

위 예제의 5~8 행에는 둘 이상의 함수 호출문이나 연산문이 하나의 문장 안에 삽입되어 있다. 이러한 문장구성이 가능한 이유는 콤마 연산자를 사용했기 때문이다.

- 연산자의 우선순위와 결합방향

순위	연산기호	연산자	결합방향
1 위	( )	함수호출	→
	[ ]	인덱스	
	- >	간접지정	
	.	직접지정	
	++ (postfix) -- (postfix)	후위증가 및 감소	
2 위	++ (prefix) -- (prefix)	전위증가 및 감소	←
	sizeof	바이트 단위 크기 계산	
	~	비트 단위 NOT	
	!	논리 NOT	
	-, +	부호 연산(음수와 양수의 표현)	
	&	주소 연산	
	*	간접지정 연산	
3 위	(casting)	자료형 변환	←
4 위	*,/,%	곱셈, 나눗셈 관련 연산	→
5 위	+, -	덧셈, 뺄셈	→

6 위	<<,>>	비트이동	→
7 위	<,>,<=,>=	대소비교	→
8 위	==, !=	동등비교	→
9 위	&	비트 AND	→
10 위	^	비트 XOR	→
11 위	|	비트 OR	→
12 위	&&	논리 AND	→
13 위	||	논리 OR	→
14 위	? :	조건연산	←
15 위	=, +=, -=, *=, /=, %=, <<=, >>=, &=, ^=, |=	대입연산	←
16 위	,	콤마연산	→

3. 키보드로부터의 데이터 입력과 C언어의 키워드

- 키보드로부터의 정수입력을 위한 scanf 함수의 호출

scanf 함수를 이용하면 키보드로부터 다양한 형태의 데이터를 입력 받을 수 있는데, 일단은 정수의 입력에 대해서만 이야기하겠다. scanf 함수의 사용방법은 printf 함수와 매우 유사하다. 다음 코드는 scanf 함수의 기본적인 사용방법을 보여준다.

```c
#include <stdio.h>

int main(void)
{
	int num;
	scanf("%d", &num); // 키보드로 입력된 정수를 변수 num에 저장하라.
}
```

키보드로부터 데이터를 입력 받으려면, 데이터의 종류에 맞게 메모리 공간을 미리 할당해야 한다. 따라서 위의 코드에서는 scanf 함수호출에 앞서 변수 num을 선언하고 있다. 그리고 이어서 scanf 함수를 호출하면서, 두 번째 인자로 num을 넘겨주고 있다.

scanf("%d", &num);

%d: 10 진수 정수형태로 입력 받아서, &num: 변수 num에 저장하라

scanf 함수에서 %d는 10 진수 정수형태의 입력을 뜻한다.

```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;
}
```

```
정수 one: 100
정수 two: 99
100+99=199
```

- 입력의 형태를 다양하게 지정할 수 있다

printf 함수를 이용해서 출력형태를 다양하게 지정하는 것이 가능하였다. 마찬가지로 scanf 함수를 이용하면 입력형태를 다양하게 지정하는 것이 가능하다. 여기서 입력형태를 다양하게 지정한다는 것은 한번의 scanf 함수호출을 통해서 여러 개의 데이터를 다양한 형태로 입력 받는 것이 가능하다는 것이다. 그리고 이렇듯 둘 이상의 데이터를 입력 받기 위해서는 아래와 같이 문장을 작성하면 된다.

```
#include <stdio.h>

int main(void)
{
        int num1, num2, num3;
        scanf("%d %d %d", &num1, &num2, &num3); // 총 3개의 10진수 정수 입력
}
```

위의 scanf 함수는 10 진수 정수형 데이터 하나, 둘, 셋을 입력 받아서 변수 num1, num2, num3 에 저장한다. 위의 scanf 함수의 호출문장에서, 서식문자의 수를 조절하여 입력 받는 데이터의 수를 조절할 수 있을 뿐만 아니라, 서식문자를 변경하여 입력 받는 데이터의 유형(형태)도 달리할 수 있다.

```
#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;
}
```

위 예제의 9 행에서는 scanf 함수호출을 통해서 총 3 개의 정수를 입력 받고 있다. 그런데 scanf 함수는 공백을 기준으로 데이터를 구분하므로, 3 개의 정수 사이에 공백에 해당하는 스페이스바, 탭 또는 엔터 키를 입력해야한다.

```
세 개의 정수 입력 : 20 88 93
20 + 88 + 93 = 201
```

4. 문제

4-1. 프로그램 사용자로부터 두 개의 정수를 입력 받아서 두 수의 뺄셈과 곱셈의 결과를 출력하는 프로그램을 작성해보자.

```
#include <stdio.h>

int main(void)
{
        int num1, num2;
        int result1, result2;

        printf("두 개의 정수를 입력하세요 : ");
        scanf("%d %d",&num1,&num2);
        result1 = num1 - num2;
        result2 = num1 * num2;

        printf("%d - %d = %d \n", num1, num2, result1);
        printf("%d * %d = %d \n", num1, num2, result2);
        return 0;
```

```
두 개의 정수를 입력하세요 : 9 2
9 - 2 = 7
9 * 2 = 18
```

4-2. 프로그램 사용자로부터 세 개의 정수 num1, num2, num3 를 순서대로 입력 받은 후에, 다음 연산의 결과를 출력하는 프로그램을 작성해보자.

num1 x num2 + num3

단, 입력 받은 세 개의 정수가 2, 4, 6 이라면 다음의 형태로 출력을 해야 한다.

2x4+6=14

```c
#include <stdio.h>

int main(void)
{
        int num1, num2, num3;
        int result;

        printf("정수 3개 입력 : ");
        scanf("%d %d %d", &num1, &num2, &num3);

        result = num1 * num2 + num3;

        printf("%d X %d + %d = %d \n", num1, num2, num3, result);
        return 0;
}
```

```
정수 3개 입력 : 2 4 6
2 X 4 + 6 = 14
```

4-3. 하나의 정수를 입력 받아서, 그 수의 제곱의 결과를 출력하는 프로그램을 작성해보자. 예를 들어서 5 가 입력되면 25 가 출력되어야 한다.

```c
#include <stdio.h>

int main(void)
{
        int num, result;

        printf("정수 하나 입력 : ");
        scanf("%d", &num);

        result = num * num;
        printf("%d ^ 2 = %d", num, result);
        return 0;
}
```

```
정수 하나 입력 : 8
8 ^ 2 = 64[lms@localhost c]$
```

4-4. 입력 받은 두 정수를 나누었을 때 얻게 되는 몫과 나머지를 출력하는 프로그램을 작성해보자. 예를 들어서 7 과 2 가 입력되면 몫으로 3, 나머지로 1 이 출력되어야 한다.

```c
#include <stdio.h>

int main(void)
{
        int num1, num2;
        int result1, result2;

        printf("정수 두 개를 입력하세요 :" );
        scanf("%d %d", &num1, &num2);

        result1 = num1 / num2;
        result2 = num1 % num2;

        printf("%d를 %d로 나눈 몫 = %d \n", num1, num2, result1);
        printf("%d를 %d로 나눈 나머지 = %d \n", num1, num2, result2);
}
```

```
정수 두 개를 입력하세요 :9 2
9를 2로 나눈 몫 = 4
9를 2로 나눈 나머지 = 1
```

4-5. 입력 받은 세 개의 정수 num1, num2, num3 을 대상으로 다음 연산의 결과를 출력하는 프로그램을 작성해보자.

(num1-num2) x (num2+num3) x (num3%num1)

```c
#include <stdio.h>

int main(void)
{
        int num1, num2, num3;
        int result;

        printf("정수 3개 입력 : ");
        scanf("%d %d %d",&num1, &num2, &num3);
        result = (num1-num2) * (num2+num3) * (num3%num1);
        printf("(%d - %d) x (%d + %d) x (%d를 %d로 나눈값의 나머지)) = %d \n", num1, num2, num2, num3, num3, num1, result);
        return 0;
}
```

```
정수 3개 입력 : 20 10 5
(20 - 10) x (10 + 5) x (5를 20로 나눈값의 나머지)) = 750
```

5. 제어문

- break 문: break 문을 만나면 지금 처리하고 있는 블록을 빠져나가 블록 이후의 문장을 처리한다. { }의 블록으로 이루어진 반복문이나 조건문 중 switch case 문 등을 빠져나가는 용도로 많이 사용된다.

- continue 문: 반복문 안에서 사용되며, 이것을 만나면 반복문의 조건을 판단하는 줄로 바로 이동한다.

- return 문: 어느곳에서나 이 구문을 만나면 해당 함수를 빠져나가게 된다.

- goto 문: 프로그래머가 지정한 줄로 실행을 이동한다. goto 문은 프로그램의 흐름을 임의로 바꾸는 것이기 때문에 코드를 알아보기 어려우며, 종종 오류발생의 원인이 된다.