함수를 중심으로 1학기 내용 복습!

관계 연산자, 논리 연산자

<대입 및 산술 연산자>

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;	→

<관계 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?	→

<논리 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)	←

<복합 대입 연산자>

조건문, 반복문

<반복문>

```
while (반복 조건)
{
    반복 내용:
}
```

```
do
{
    반복 내용:
} while (반복 조건);
```

```
for (초기식; 조건식; 증감식)
{
    반복 내용:
}
```

<조건문>

```
if (비교, 논리 연산)
{
    괄호 속 연산이 참일 때 수행:
}
```

```
if (조건1)
{조건1이 참일 경우}
else if (조건2 )
{조건2가 참일 경우}
else
{나머지 경우}
```

```
switch (n)
{
    case 1:
        실행문1; break;
    case 2:
        실행문2; break;
    default:
        나머지 실행문;
}
```

3

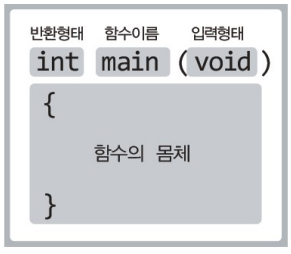
함수를 만드는 이유

"Divide and Conquer!"

다수의 작은 단위 함수를 만들어서 프로그램을 작성하면 큰 문제를 작게 쪼개서 해결하는 효과를 얻을 수 있다.
그러나 함수를 만드는 이유 및 이점은 이보다 훨씬 다양하다.

main 함수를 포함하여 함수의 크기는 작을수록 좋다. 무조건 작다고 좋은 것은 아니지만, 불필요하게 큰 함수가 만들어지지 않도록 주의해야 한다.

하나의 함수는 하나의 일만 담당하도록 디자인 되어야 한다. 물론 하나의 일이라는 것은 매우 주관적인 기준이다. 그러나 이러한 주관적 기준 역시 프로그래밍에 대한 경험이 쌓이면 매우 명확한 기준이 된다.



4

함수의 입력과 출력: printf 함수도 반환을 합니다.

```
int main(void)
{
    int num1, num2;
    num1=printf("12345\n");
    num2=printf("I love my home\n");
    printf("%d %d \n", num1, num2);
    return 0;
}
```

printf 함수도 사실상 값을 반환한다. 다만 반환값이 필요 없어서 반환되는 값을 저장하지 않았을 뿐이다.
printf 함수는 출력된 문자열의 길이를 반환한다.

```
12345
I love my home
6 15
```

실행결과

함수가 값을 반환하면 반환된 값이 함수의 호출문을 대체한다고 생각하면 된다.
예를 들어서 아래의 printf 함수 호출문이 6을 반환한다면,

num1 = printf ("12345\n");

함수의 호출결과는 다음과 같이 되어 대입연산이 진행된다.

num1 = 6;

5

함수의 구분

유형 1: 전달인자 있고, 반환 값 있다! 전달인자(○), 반환 값(○)
유형 2: 전달인자 있고, 반환 값 없다! 전달인자(○), 반환 값(×)
유형 3: 전달인자 없고, 반환 값 있다! 전달인자(×), 반환 값(○)
유형 4: 전달인자 없고, 반환 값 없다! 전달인자(×), 반환 값(×)

전달인자와 반환 값의 유무에 따른 함수의 구분!

6

전달인자 반환 값 모두 있는 경우

전달인자는 int형 정수 둘이며, 이 둘을 이용한 덧셈을 진행한다.

덧셈결과는 반환이 되며, 따라서 반환형도 int형으로 선언한다.

마지막으로 함수의 이름은 Add라 하자!

```
A.  B.  C.
int Add (int num1, int num2)
{
    int result = num1 + num2;
D.  return result;
}
```

A. 반환형
B. 함수의 이름
C. 매개변수
D. 값의 반환

함수호출이 완료되면 호출한 위치로 이동해서 실행을 이어간다.

```
int Add(int num1, int num2)
{
    return num1+num2;
}
int main(void)
{
    int result;
    result = Add(3, 4);
    printf("덧셈결과1: %d \n", result);
    result = Add(5, 8);
    printf("덧셈결과2: %d \n", result);
    return 0;
}
```

덧셈이 선 진행되고 그 결과가 반환됨

실행결과

```
덧셈결과1: 7
덧셈결과2: 13
```

7

전달인자나 반환 값이 존재하지 않는 경우

```
void ShowAddResult(int num)   // 인자전달 (O), 반환 값 (X)
{
    printf("덧셈결과 출력: %d \n", num);
}
```

```
int ReadNum(void)   // 인자전달 (X), 반환 값 (O)
{
    int num;
    scanf("%d", &num);
    return num;
}
```

```
void HowToUseThisProg(void)   // 인자전달 (X), 반환 값 (X)
{
    printf("두 개의 정수를 입력하시면 덧셈결과가 출력됩니다. \n");
    printf("자! 그럼 두 개의 정수를 입력하세요. \n");
}
```

8

4가지 함수 유형을 조합한 예제

```
int Add(int num1, int num2)    // 인자전달 (O), 반환 값 (O)
{
    return num1+num2;
}

void ShowAddResult(int num)    // 인자전달 (O), 반환 값 (X)
{
    printf("덧셈결과 출력: %d \n", num);
}

int ReadNum(void)    // 인자전달 (X), 반환 값 (O)
{
    int num;
    scanf("%d", &num);
    return num;
}

void HowToUseThisProg(void)    // 인자전달 (X), 반환 값 (X)
{
    printf("두 개의 정수를 입력하시면 덧셈결과가 출력됩니다. \n");
    printf("자! 그럼 두 개의 정수를 입력하세요. \n");
}
```

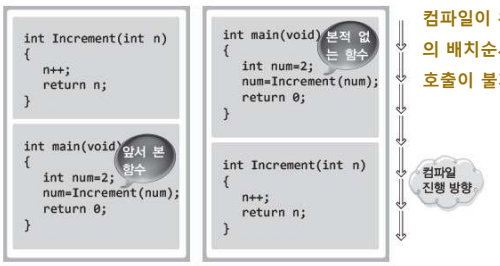
```
int main(void)
{
    int result, num1, num2;
    HowToUseThisProg();
    num1=ReadNum();
    num2=ReadNum();
    result = Add(num1, num2);
    ShowAddResult(result);
    return 0;
}
```

Review_1st_Semeter1.c

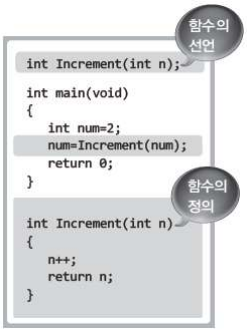
실행결과

```
두 개의 정수를 입력하시면 덧셈결과가 출력됩니다.
자! 그럼 두 개의 정수를 입력하세요.
12 24
덧셈결과 출력: 36
```


함수의 정의와 그에 따른 원형의 선언



컴파일이 위에서 아래로 진행이 되기 때문에 함수의 배치순서는 중요하다. 컴파일 되지 않은 함수는 호출이 불가능하다.



이후에 등장하는 함수에 대한 정보를 컴파일러에게 제공해서 이후에 등장하는 함수의 호출문장이 컴파일 가능하게 도울 수 있다.
이렇게 제공되는 함수의 정보를 가리켜 '함수의 선언'이라 한다.

```
int Increment(int n);   // 함수의 선언
int Increment(int);   // 위와 동일한 함수선언, 매개변수 이름 생략 가능
```

다양한 종류의 함수 정의1

```
int main(void)
{
    printf("3과 4중에서 큰 수는 %d 이다. \n", NumberCompare(3, 4));
    printf("7과 2중에서 큰 수는 %d 이다. \n", NumberCompare(7, 2));
    return 0;
}

int NumberCompare(int num1, int num2)
{
    if(num1>num2)
        return num1;
    else
        return num2;
}
```

중간에도 얼마든지 return문이 올 수 있다.

실행결과

```
3과 4중에서 큰 수는 4 이다.
7과 2중에서 큰 수는 7 이다.
```

```
printf("3과 4중에서 큰 수는 %d 이다. \n", NumberCompare(3, 4));
printf("7과 2중에서 큰 수는 %d 이다. \n", NumberCompare(7, 2));

printf("3과 4중에서 큰 수는 %d 이다. \n", 4);
printf("7과 2중에서 큰 수는 %d 이다. \n", 7);
```

위의 두 문장한 NumberCompare 함수호출 이후 왼쪽과 같이 된다.

11

다양한 종류의 함수 정의2

```
int AbsoCompare(int num1, int num2);   // 절댓값이 큰 정수 반환
int GetAbsoValue(int num);     // 전달인자의 절댓값을 반환

int main(void)
{
    int num1, num2;
    printf("두 개의 정수 입력: ");
    scanf("%d %d", &num1, &num2);
    printf("%d와 %d중 절댓값이 큰 정수: %d \n",
        num1, num2, AbsoCompare(num1, num2));
    return 0;
}
int AbsoCompare(int num1, int num2)
{
    if(GetAbsoValue(num1) > GetAbsoValue(num2))
        return num1;
    else
        return num2;
}
int GetAbsoValue(int num)
{
    if(num<0)
        return num * (-1);
    else
        return num;
}
```

```
if(GetAbsoValue(num1) > GetAbsoValue(num2))
    . . . .

if( 5 > 9 )
    . . . .
```

GetAbsoValue 함수호출 이후

이 예제에서 보이듯이 함수의 호출문장은 어디에든 놓일 수 있다.

Review_1st_Semeter2.c

실행결과

```
두 개의 정수 입력: 5 -9
5와 -9중 절댓값이 큰 정수: -9
```

12

문자의 입출력 실습 (연습 1)

- 문자 입력 및 변환 출력

```
알파벳 소문자 입력 : m
입력한 알파벳 소문자 m의 대문자는 M
```

  - 소문자 입력 : scanf("%c", &a);
  - 대문자로 변환 : a – 32
  - 대문자 출력

13

정수의 입출력 및 for문 활용 실습 (연습 2)

- 정수 값의 입력 및 합을 계산하여 출력

```
숫자를 입력하세요 : 2
2부터 4까지의 합계는 9입니다.
숫자를 입력하세요 : -3
-6부터 -3까지의 합계는 -18입니다.
숫자를 입력하세요 : 0
수고하셨습니다.
```

  - 정수 값 입력 : scanf("%d", &n);
  - n부터 2n까지의 합을 계산 : for문 사용
  - 반복 수행 : while문 사용
  - 0 값 입력 시 출력 후 종료

14

새로운 함수를 활용하여 정수 값 중 큰 값 찾기 실습 (연습 3)

- 3개의 정수 값 중 최대 값을 max_int라는 함수에서 찾은 후 결과 값을 main 함수에서 출력

```
세 개의 정수 중 가장 큰 수를 출력합니다.

첫 번째 입력 정수는 : 30
두 번째 입력 정수는 : 10
세 번째 입력 정수는 : 20

입력한 3개의 정수 30, 10, 20 중 가장 큰 정수는 30입니다.
```

  - 3개의 정수를 입력 받음 :
  - 정수들을 max_int 함수로 보내 최대 값을 찾아서 보내도록 함
  - max_int 함수에서 보내온 최대 값을 main 함수에서 출력함

15

새로운 함수를 활용하여 공배수 값 찾아 출력 실습 (연습 4)

- 1개의 정수 값에 대해 2와 3의 공배수를 모두 출력

```
1보다 큰 정수 하나를 입력하세요 : 50
1부터 50까지의 숫자 중 2와 3의 공배수는
6 12 18 24 30 36 42 48이며
개수는 8개입니다.
```

  - 1개의 정수를 입력 받음 : scanf("%d", &a);
  - 입력 받은 정수 값 (예, a)을 co_multi 함수로 보내 1부터 a까지의 정수들 중 2와 3의 공배수를 모두 출력하도록 함 : for문 사용 (1부터 a까지 loop)

16

점수 2개를 입력 받아 과락 여부 판정 실습 (연습 5)

- 2개의 점수 중 과락이 있는지, 평균이 기준 이하인지를 판정하여 그 결과를 출력

```
필기 점수를 입력하세요 : 60
실습 점수를 입력하세요 : 85
과락이 없고 평균이 72.5점이므로 "합격"입니다.
```

```
필기 점수를 입력하세요 : 55
실습 점수를 입력하세요 : 85
55점으로 과락이므로 "불합격"입니다.
```

  - 2개의 점수를 입력 받음 : scanf 사용
  - 점수를 success_fail 함수를 넘겨서 판정토록 함
  - success_fail 함수에서 판정 후 그 결과를 main 함수로 보냄
  - main 함수에서 최종 결과를 출력

17

1학기 복습이 끝났습니다. 질문 있으신지요?