7. 문장(statement) vs. 표현식(expression)

• if문은 괄호 안에 명시된 조건을 비교하여 연관된 문장의 실행 여부를 결정합니다.

```
if (i>j) printf("i is greater than j");
```

• 위 문장은 괄호 안에 명시된 i>j가 만족된다면 printf()를 실행하고, i>j가 만족되지 않는다면 printf()를 실행하지 않습니다.

• 우리가 유의해야할 사항은 if는 다른 문장을 포함한 1개의 문장이라는 것입니다. 즉, 위의 문장은 1개의 문장, if-문장 입니다.

- if-문은 다음과 같은 문법을 가집니다.

```
if (비교문장1)
  문장1;
[else if (비교문장2)
  문장2;][...]
[else
  문장3;]
```

- 아래 프로그램의 출력 결과는 얼마일까요?

```
#include <stdio.h>

void main() {
    int i=2,j=3;

    if (i>j)//①
        i=i+1;
        if (i<j)//②
            j=j+1;
    else//③
        printf("%d\n",i);
}
```

"③번의 else문이 ②번 if와 짝 지워진 것인지, ①번 if와 짝 지워진 것인지 어떻게 알 수 있는가요?"

- 이제 소스를 아래와 같이 배열해 봅시다.

```
#include <stdio.h>

void main() {
    int i=2,j=3;

    if (i>j)//(1)
        i=i+1;
    if (i<j)//(2)
        j=j+1;
    else//(3)
        printf("%d\n",i);
}
```

- **달랑거리는 else문제(dangling else problem)**

  "else의 위쪽으로 봐서, else와 짝 지워지지 않은 가장 가까운 if 혹은 else if와 짝 짓습니다."

블록(block)이 뭐죠?

C/C++ 언어의 구조에 의해 여러 개의 독립된 문장을 1개의 문장 취급해야할 필요성이 빈번하게 생깁니다. 구조적 언어structured programming language는 이러한 기능을 위해서 블록block 기능을 제공합니다. 예를 들면 Pascal 언어에서는 블록을 begin...end로 표현합니다. C언어는 블록을 나타내기 위해 {...}를 사용합니다. 즉, {와 } 사이에 명시된 두개 이상의 문장들은 한 개의 문장으로 취급됩니다. 블록은 실제로 몇가지 추가적인 특징을 가집니다. 이러한 사항은 후에 자세하게 설명하겠습니다.

```
#include <stdio.h>

void main() {
    int i=2,j=3;

    if (i>j) {//①
        i=i+1;
        if (i<j)//②
            j=j+1;
    } else//③
        printf("%d\n",i);
}
```

- **문장statement**과 **표현식expression**을 명확히 구분할 필요가 있습니다.
- 문장 자체의 값이 C에서 지원하는 데이터 값 - 일반적으로 정수 - 인 경우, 이러한 문장을 표현식이라고 합니다.
- 표현식은 문장입니다. 하지만, 문장은 표현식이 아닙니다.

```
int i=2,j=3;//(1)
i+j;//(2)
i=2+3;//(3)
i=j=2;//(4)
return i;//(5)
```

- i=j=2;라는 문장의 의미는 다음과 같습니다.

"2를 j에 대입하고, *j=2*를 i에 대입합니다. 물론 *j=2*는 2입니다."

- if문의 문법을 정확하게 새로 적어 보겠습니다.

```
if (표현식1)
  문장1;
[else if (표현식2)
  문장2;][...]
[else
  문장3;]
```

- 위와 같은 if-문의 구조에서, if 괄호 안의 표현식이 '참이다', '거짓이다'의 판단기준은 무엇일까요?
- C에는 참/거짓이란 값은 존재하지 않습니다.

(1) 숫자 표현식 0은 거짓입니다.
(2) 0이외의 모든 표현식은 참입니다.

• 컴파일러는 **관계 연산relational operation**의 결과를 판단하기 위해, i>j에 대해, i-j를 수행합니다.
• 결과를 검사하여, 모든 비트가 0이면, i>j를 0으로 평가합니다. 그렇지 않으면, i>j를 1로 평가합니다. 즉, i>j라는 문장은 결과가 0혹은 1인 표현식인 것입니다.

```
if (i>j)
    m=1;
else
    m=0;
```

• 관계 연산자의 특징을 이용하면, 다음과 같이 문장을 간단히 쓸 수 있습니다.

```
m=(i>j);
```

• i가 j보다 크면 1을 리턴하는 함수를 만든다고 가정해 봅시다. 역시 다음과 같은 문장을 사용하여 함수를 구성할 수 있습니다.

```
return i>j;
```

- 아래 프로그램의 출력결과는 무엇일까요?

```
#include <stdio.h>

void main() {
    int i=2,j=3,k=4;

    printf("%d,%d,%d,%d\n", i>j, i==j, j!=k, i<=k);
}
```

문장의 종류

■ 라벨문(labeled statement) : switch에서 혹은 goto의 대상으로 사용됩니다.

■ 표현식(expression statement): 문장의 결과가 값value입니다. 대부분의 수학 문장과 몇 특수한 문장을 포함합니다.

■ 복합문(compound statement) : 블록을 의미합니다.

■ 선택문(selection statement) : if, switch문이 있습니다.

■ 반복문(iteration statement) : for,while,do...while문이 있습니다.

■ 분기문(jump statement): break, continue와 goto문이 있습니다. goto문은 이 책에서 설명하지 않습니다.

■ 선언문(declaration statement) : 변수선언, 함수선언문 등이 있습니다. 다른 선언문은 앞으로 배우게 될 것입니다.

■ try-블록문(try-block statement): '만화가 있는 C++'에서 자세히 다룹니다.

실습문제

1. 대입문은 표현식이기는 하지만, 함수의 파라미터로 사용하는 것은 (가능하지만) 추천되지 않습니다. 이유가 무엇일까요? 구체적인 예를 들어 설명하세요(힌트: 호출 관례calling convention).

2. 아래 문장의 출력결과가 나오는 과정을 상세하게 설명하세요.

```
printf("hello%d",printf("abc"));
```

@