연산자 조건문 반목문

멘토 : 이승준 멘티 : 홍우기, 김도영, 노명욱

연산자(OPERATOR)

복합대입 연산자 += -= …

전위 후위 연산자 ++ - -

논리 연산자 && ||

비교 연산자 > < >= <= == !=

삼항 연산자 ? :

복합대입 연산자

대입 연산자 : = 좌측 변수의 주소에 우측 상수 혹은 우측 변수의 상수를 대입한다.

```
num = 10;
num2 = num;
```

복합 대입 연산자 : 아래 식을 줄여서 쓴 연산자이다.

```
num = num + 10; >>>num = 20
```

(연산자 우선순위에 의해 num+10 이 먼저 일어나고 그 다음 대입 연산자에 의해 num에 20이라는 값이 들어간다.)

[num = num + 10] 을 [num += 10] 으로 줄인 것이다.
+= : 자기 자신의 값 + 왼쪽 상수값을 자기 자신에 대입한다.

전위 연산자, 후위 연산자

num += 1을 쓰는 경우가 매우 많기 때문에 또 한 번 더 줄인 연산자이다.

단, ++ 혹은 - - 를 변수의 앞뒤 어느쪽에 쓰냐에 따라 달라진다.

전위 연산자 : ++num >> num에 바로 num += 1

후위 연산자 : num++ >> 이 연산자 다음 줄부터 num += 1

	++	- -
전위	++num 사실상 num+=1	- - num 사실상 num-=1
후위	num++ 다음 줄부터 num+=1적용	num- - 다음 줄부터 num-=1적용

논리 연산자

X	Y	&&	||	!X
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

boolean 이라는 자료형에는 true(참) false(거짓) 이 두 상수 밖에 존재하지 않는다.

true : 숫자로는 0이 아닌 모든 수(1로 생각하자)

false : 0

비교 연산자

수학에서 하던 > < >= <= 그냥 이거다. 같다 다르다 를 == 과 != 으로 표현한다.

X	Y	Z		X < Y	X <= Z	X == Y	X != Y
10	20	10		true	true	false	true

삼항 연산자

조건과 대입을 혼합한 연산자

변수 = (조건문) ? 값1(참 일때 대입되는 값) : 값2 (거짓 일때 대입되는 값)

num = (num1 > num2) ? 10 : 20;

>> (true) => num = 10
>> (false) => num = 20

반복문(LOOP)

while

do while

for

break & continue

while

```
while( 조건문) {
   반복 영역
}
```

(조건문)이 참이면 {반복 영역}을
한번 실행 한 뒤 다시 while로 돌아간다

여기서 num 이 반복적으로 +1 됨으로써
결국 6이 되었을 때 (조건문은) false
따라서 while 문은 종료된다.

```c
#include <stdio.h>

int main (void){

  int num = 1;
  while(num < 6){
    printf("num is %d\n",num);
    num++; //후위 연산자
  }

  return 0;
}
```

```
num is 1
num is 2
num is 3
num is 4
num is 5
```

while

무한 반복 (⚠️우리가 피해야 하는 상황⚠️)

while 의 원리는 매우 단순하다.
(조건문) 이 true면 실행하고 false면 끝낸다.
무한 반복은 코드가 반복 될 때
(조건문)이 영원히 참이면 발생한다.

while(1){} 을 break; 를 이용하는 방법도 있다.

```c
#include <stdio.h>

int main (void){

  while(1){
    printf("print 1");
  }
```

```
 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1p
rint 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1prin
t 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1
print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1pri
nt 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print
1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1pr
int 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print
 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1p
rint 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1prin
t 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1
print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1pri
nt 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print
1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1pr
int 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print
 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1p
rint 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1prin
t 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1
print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1pri
nt 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print
1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1pr
int 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print
 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1p
rint 1print 1print 1print 1print 1print 1print 1print 1print 1print 1print 1prin
```

do while

최소한 1회 이상 실행 되어야 하는 반복을 쓰고 싶을때
(반복 영역이 조건식의 검사보다 우선일 때)

```
do {
   반복 영역
} while (num != 0);
```

```
정수 입력(0 to quit) : 5
정수 입력(0 to quit) : 4
정수 입력(0 to quit) : 3
정수 입력(0 to quit) : 2
정수 입력(0 to quit) : 1
정수 입력(0 to quit) : 0
합계 : 15 logout
```

```c
#include <stdio.h>

int main (void){
  int num = 0, sum = 0;
  do{
    printf("정수 입력(0 to quit) : ");
    scanf("%d",&num);
    sum += num;
  }while(num != 0);
  printf("합계 : %d ",sum);
  return 0;
}
```

for

for 문은 while 문에서 자주 쓰이는 형식을 보기 쉽게 정리한 식이다. (while > for)

초기식
while (조건식){
  반복 실행문
  증감식
}

```c
#include <stdio.h>

int main (void){

  int num = 1;
  while(num < 6){
    printf("num is %d\n",num);
    num++; //후위 연산자
  }

  return 0;
}
```

for(초기식; 조건식; 증감식){
  반복 실행문

}

```c
#include <stdio.h>

int main (void){

  for(int i = 1; i<6; i++){
    printf("num is %d\n",num);
  }

  return 0;
}
```

break & continue

break 는 현재 가장 가까운 반복문을 탈출(끝내는)하는 코드이다.
continue 는 현재 가장 가까운 반복문을 계속 해서 진행하라는 코드이다.

반복문과 조건문을 같이 쓸때 자주 쓰인다.

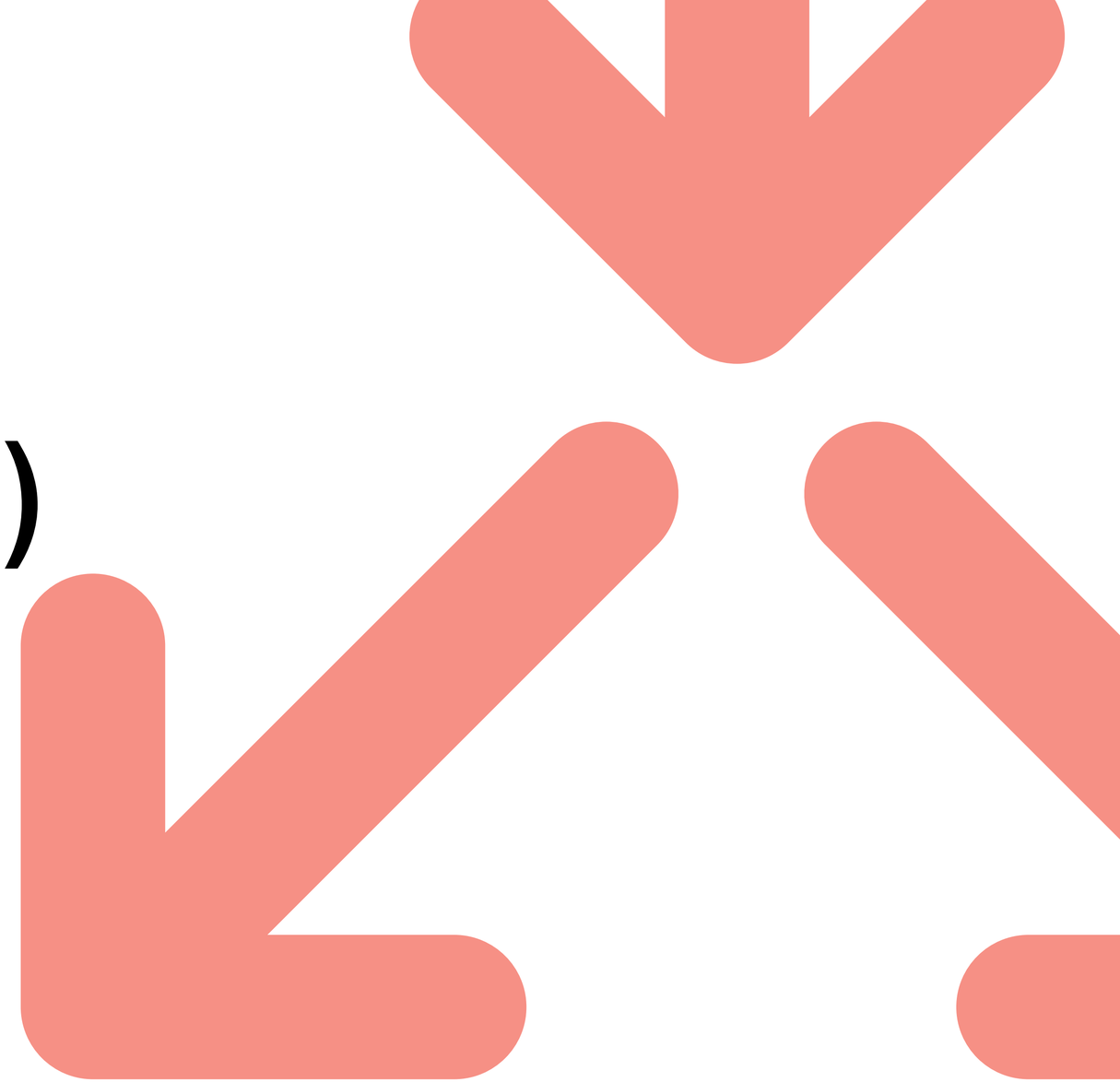
조건문(CONDITION)

if

else if

else

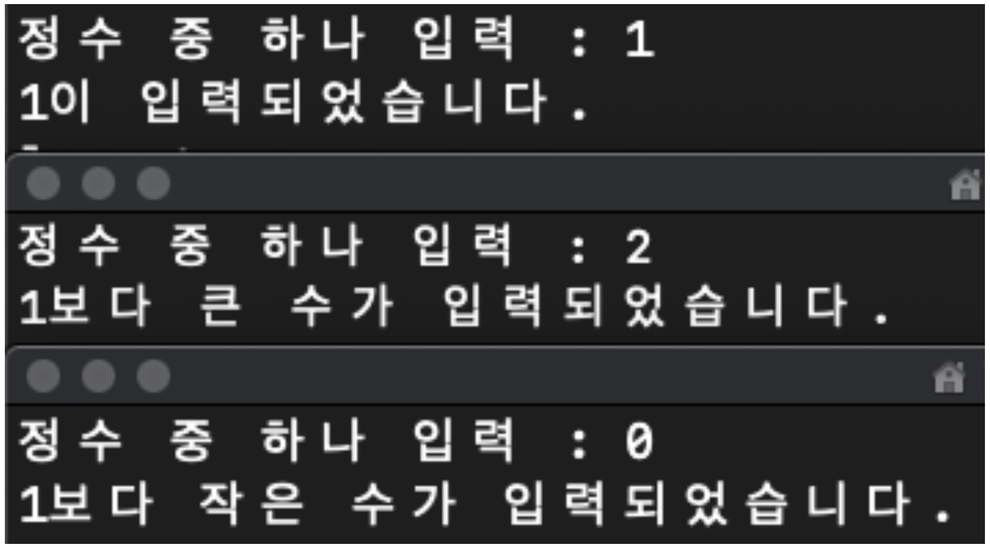
switch



if

if 문이 필요한 이유는 프로그램을 구현하다 보면 상황(조건)에 따라 선택적으로 실행해야 하는 영역도 존재한다.

입력 된 수에 따라서 다른 결과가 나온다.




```c
#include <stdio.h>

int main (void){
  int num;
  printf("정수 중 하나 입력 : ");
  scanf("%d",&num);
  if(num == 1){
    printf("1이 입력되었습니다.\n");
  }
  if(num > 1){
    printf("1보다 큰 수가 입력되었습니다.\n");
  }
  if(num < 1){
    printf("1보다 작은 수가 입력되었습니다.\n");
  }

  return 0;
}
```

키워드 (조건문) {실행문}

else if

만약 1억개의 if 문이 있다고 가정하자,
num의 값에 따라 if(num == 1) 에서 if(num == 1억)까지 서로 다른 실행문이 있다고 가정하자
(다음 페이지 예시 코드 참조)

만약 내가 입력한 수가 10 이면 컴퓨터는 >>10이 입력한 경우를 실행!
그런데 컴퓨터 에서는 나머지 9999990 번의 if 문의 조건문을 검토하게 될 것이다.

왜냐하면 컴퓨터는 계속 코드를 아래로 읽는 것이 원칙이기 때문이다.

이는 매우 비효율적이다. 왜냐하면 당연히 num이 동시에 같은 숫자 일 수 없기 때문에
굳이 검토하지 않아도 나머지 if 문은 flase 임을 알 수 있기 때문이다

우리는 else if로 여러가지 else if 중 하나가 참이면 나머지 아래의 else if는 실행 해주지 않아도 된다는 사실을 컴퓨터에게 명령을 내릴 수 있다.

바로 이 점이 else if 문을 써줘야 하는 이유이다. 효율성

else if

```c
#include <stdio.h>

int main (void){

  if(num == 1){
    ...
  }
  if(num == 2){
    ...
  }
  .
  .
  .
  if(num == 99999999){
    ...
  }
  if(num == 100000000){
    ...
  }

  return 0;
}
```

```c
#include <stdio.h>

int main (void){

  if(num == 1){
    ...
  }
  else if(num == 2){
    ...
  }
  .
  .
  .
  else if(num == 99999999){
    ...
  }
  else if(num == 100000000){
    ...
  }

  return 0;
}
```

다시 설명하자면 좌측은 중간에 10이 나와도 1억까지 조건문을 살피고

우측 코드는 중간에 10 이 나오면 다음 연결 되는 모든 if else 문들은 그냥 건너 뛴다.

else if 는 무조건 본인 이전에 if 가 있어야 하며
그 if 가 기준이 된다.

else

if, else if … 중 단 하나도 만족하는 조건이 없으면 무조건! else 문의 실행문이 실행된다.

else 문에는 조건문이 없다.

추가) else 는 가장 가까운 if 혹은 else if 에 대해서 else 가 발동된다.

주의) else문 앞에 if 혹은 else if 가 없거나 또다른 else 가 있으면 에러가 발생한다.

```c
#include <stio.h>

int main (void){

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

  if(num == 1){
    printf("1이다.\n", );
  }
  else{
    printf("1이 아니다.\n", );
  }

  return 0;
}
```

```
7
1이  아 니 다 .
```

switch 문

if else 의 조건문이 모두 ‘==‘ 동등 비교문인 경우 더 간결하게 바꿀 수 있다.

```c
#include <stdio.h>

int main (void){

  if(n == 1){
    statement1;
  }
  else if(n == 2){
    statement2;
  }
  else if(n == 3){
    statement3;
  }
  .
  .
  .


  return 0;
}
```

```c
#include <stdio.h>

int main (void){

  switch(n){
    case 1 :
      statement1;
      break;
    case 2 :
      statement2;
    case 3 :
      statement3;
      .
      .
      .
    default :
      ...
  }
  return 0;
}
```

들여쓰기 (tab)

가독성을 위한 들여쓰기(tab 키)

if, else if, else, switch, while, do while, for 문 안의 실행문의 가독성을 위해서

실행문은 반드시 들여쓰기 하는 것을 원칙으로 한다.