

# Control Flow

프로그래밍 입문(2)

# This Week

- 이번 주에는 과제가 있습니다.
- 실습대체 과제가 아닌 여러분이 문제를 푸는 (간단한 프로그램 작성하는) 과제입니다.
- 실습은 각자 진행하시고, 과제 제출로 출석을 대신합니다.

# Topics

- 제어문 (Control Statement)
  - 분기문: **if, switch**
  - 반복문 (loop): for, while, do-while

# 제어문이란?

- 프로그램 실행의 흐름을 제어하기 위해 사용하는 문장들.
- 프로그램의 **특정 부분을 특정 조건에만 실행하고 싶다.**
  - 분기문을 사용.
- 프로그램의 **특정 부분을 반복해서 실행하고 싶다.**
  - 반복문을 사용.

# 분기문 if

- 가장 기본적인 분기문은 if문으로, 다음의 두 가지 형태로 사용됩니다.
- 특정 조건<condition> 만족시 실행할 문장이 1개인 경우.
  - `if (<condition>)`  
    <statement>;
- 특정 조건 만족시 실행할 문장이 1개 이상인 경우.
  - `if ( <condition> ) {`  
    <statement\_1>;  
    ...  
    <statement\_n>;  
}

# 특정 조건?

- 프로그램에서 조건을 표현할 방법이 필요합니다.
- 프로그램의 입장에서, 또는 분기문/반복문의 입장에서,
  - 중요한 것은 조건을 평가한 것이 참(true)이나 거짓(false)이나 뿐입니다.
- **bool** 데이터형의 값을 가질 수 있는 표현은 어떤 것이든 사용이 가능합니다.

# 조건식으로 사용가능한 것들

- Boolean Literal: `true`, `false`
  - 반복문에서 주로 사용함.
- `bool` 데이터형 변수: `bool a;`
  - 조건문, 반복문에서 사용함. Switch처럼 쓰는 경우.
- `bool`형 값으로 평가되는 표현식: `a < b`, `a == b`
  - 관계/비교 연산자를 사용한 표현식이 논리 연산자를 이용하여 연결됨.
- `bool`형 값을 반환하는 함수: `bool isValid(string phone_num);`
- 언제나 `bool`형의 값 대신 `int`형 값이 0인지 아닌지로 `true`, `false`를 대신할 수 있음.

# if-else문

- else를 사용하여 if의 조건식이 거짓이었을 경우 실행할 부분을 추가할 수 있음.
- ```
if (<condition>)  
    <statement1>;  
else  
    <statement2>;
```
- ```
if ( <condition> ) {  
    <statement_1>;...<statement_n>;  
} else {  
    <statement_1>;...<statement_n>;  
}
```



# 프로그램의 흐름

- <condition>을 평가.
- 참이라면 (1)을, 거짓이라면 (2)를 실행.

```
if ( <condition> ) {  
    <statement_1>;  
    ...  
    <statement_n>;  
} else {  
    <statement_1>;  
    ...  
    <statement_n>;  
}
```

(1)

(2)

# if의 중첩

- else문에서 실행할 문장을 다시 if문으로 할 수 있음.
- 다음의 둘은 하는 일이 동일한 코드임 - { }로 묶느냐 마느냐 차이.

```
if ( <condition1> ) {  
    <statement_1>;  
} else if ( <condition2> ) {  
    <statement_2>;  
} else {  
    <statement_3>;  
}
```

```
if ( <condition1> ) {  
    <statement_1>;  
} else {  
    if ( <condition2> ) {  
        <statement_2>;  
    } else {  
        <statement_3>;  
    }  
}
```

# 프로그램의 흐름

- <condition1>을 평가.
- 참이라면 <statement\_1>을, 거짓이라면 else로 넘어감.
- <condition2>를 평가.
- 참이라면 <statement\_2>를,
- 거짓이라면 <statement\_3>를 실행.

```
if ( <condition1> ) {  
    <statement_1>;  
} else {  
    if ( <condition2> ) {  
        <statement_2>;  
    } else {  
        <statement_3>;  
    }  
}
```

# 삼항 조건식

## Ternary Conditional Expression

- 조건을 평가한 후 참/거짓에 따라 값이 달라지는 경우에만 쓸 수 있는 if-else 대체 표현.
- `int a;`
- `if (<condition>)  
    a = 5;  
else  
    a = a / 2;`
- `int a = <condition> ? 5 : a / 2;`

# 조건식 평가방법

- 비교/관계 연산자 사용한 표현식들.
  - 그 의미에 맞게 참, 거짓을 평가합니다.
  - e.g.)  $a \geq b$ ,  $i < \text{size}$ ,  $\text{str.size()} > 0$
- 여러 개의 표현식은 논리 연산자로 연결됩니다.
  - e.g.)  $a > 0 \ \&\& \ b > 0$

# 논리연산자

- 논리 연산자는 우리가 일반적으로 알고 있는 논리 연산을 표현.
- 다음의 세 종류가 있습니다.
  - AND: &&
    - $a > 0 \ \&\& \ b > 0$ : a가 0보다 크고, b도 0보다 크면 참.
  - OR: ||
    - $a > 0 \ || \ b > 0$ : a가 0보다 크거나, b가 0보다 크면 참.
  - NOT: !
    - $!(a > 0)$ : a가 0보다 크지 않으면 참. 뒤에 나오는 표현식의 값을 뒤집는 효과.

# Short Circuiting

- 효율적인 프로그램 실행을 위해, 논리 연산자로 연결된 앞의 표현식 값에 따라 뒤의 표현식 평가를 생략하는 방법.
- e.g.) `int a = 10, b = 5;`
  - `a < 0 && b > 0` → `a < 0`은 거짓이므로, 뒤의 값이 무엇이든 이 표현식은 거짓이 됨.
  - `a > 0 || b > 0` → `a > 0`은 참이므로, 뒤의 값과 상관없이 이 표현식은 참이 됨.

# Short Circuiting

- 뒤 부분이 평가되지 않는다는 의미?
- 조건식에 나타난 코드가 다른 효과를 갖는 경우.
  - $a > 0$ 이 거짓이므로, `update()`는 아예 실행되지 않음.
- `update()`는 `global`의 값을 1로 바꾸는 일을 하지만, `global`을 출력했을 때 10이 나오게 됨.
- 이런 형태의 코드를 작성할 때 신중하게 이 효과를 고려해야 함.

```
int global = 10;
int update() {
    global = 1;
    return 1;
}
```

```
a = -1;
if(a > 0 && update() > 0) {
    cout << "condition is true" << endl;
} else {
    cout << "condition is false" << endl;
}
cout << "global = " << global << endl;
```



# if문 Tips

- bool형 변수를 스위치처럼 사용할 수 있음.
  - `bool printLog = false; //disabled.`
  - ```
...  
if (printLog) {  
    cout << "log message" << endl;  
}  
...
```
- 스위치가 켜진 경우에만(`printLog=true`) 프로그램의 특정 부분이 실행됨.

# if문 Tips

- 조건식에 대입문을 사용할 수 있음.
- `if ((a = b - c) > 0) {  
    <statement>  
}`
- 빨간 부분은 `b - c`값을 갖고, 이 값을 `a`에도 대입함.
- 비록 이런 방식은 사용이 가능한 하지만, 최대한 사용하지 않는 것이 좋습니다.

# switch문

- 변수<var>의 값에 따라 경우가 나누어질 때 사용.
- switch로 변수를 지정하고, case로 정수값을 지정.
- 변수의 값이 case의 정수와 일치하면 그 위치로 점프.
- default는 case로 지정한 값이 나타나지 않았을 경우 실행.

```
switch(<var>) {  
    case <int1>:  
        <statement1>;  
        break;  
    case <int2>:  
        <statement2>;  
        break;  
    default:  
        <statement3>;  
}
```

<var> == <int1>

<var> == <int2>

# switch와 if-else

- 실제로 아래의 두 코드는 동일한 동작을 하는 코드입니다.

```
switch(<var>) {  
    case <int1>:  
        <statement1>;  
        break;  
    case <int2>:  
        <statement2>;  
        break;  
    default:  
        <statement3>;  
}
```

```
if(<var> == <int1>)  
    <statement1>;  
else if(<var> == <int2>)  
    <statement2>;  
else  
    <statement3>;
```

# break문

- break문은 실행을 중단하고 현재의 switch문을 빠져나오게 함.
- switch문에서는 값이 일치하는 case의 위치로 그저 '점프'할 뿐임.
  - 점프한 이후 그 아래 있는 모든 코드를 실행함.
- break로 적절하게 코드를 빠져나올 필요가 있음.
- 가장 마지막 부분 break는 생략.

```
switch(<var>) {  
    case <value1>:  
        <statement1>;  
        break;  
    case <value2>:  
        <statement2>;  
        break;  
    default:  
        <statement3>;  
}
```

# case에 사용가능한 표현식

- 정수형의 상수값으로 평가되는 것이면 무엇이든 가능.
  - 정수 literal: 1, 2, 3, ... → case 1:
  - 정수형의 상수: `const int APPLE = 1;`  
case APPLE:
  - 정수형 상수로 평가되는 표현식: case 2+3:

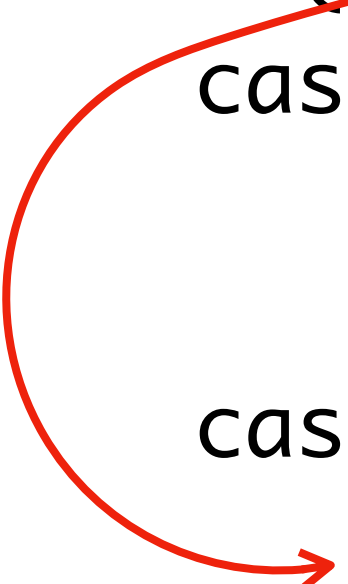
# 언제 switch를 사용해야 하는가?

- 경우의 수가 다양한 경우.
  - 특정 변수의 값에 따라 프로그램의 실행이 달라지고, if-else로 나타내기에 경우의 수가 많은 경우.
- 코드를 훨씬 간결하게 표현 가능하고, 나중에 경우를 추가하기도 편함.
- 경우의 수가 많아지면 if-else를 사용하면 비교 횟수가 늘어나게 됨.

# switch와 if-else

<var>값에 따라 바로 점프

```
switch(<var>) {  
    case <int1>:  
        <statement1>;  
        break;  
    case <int2>:  
        <statement2>;  
        break;  
    default:  
        <statement3>;  
}
```



<statement2>를 실행하는 경우  
→ 2번의 비교가 일어남.

```
if(<var> == <int1>)  
    <statement1>;  
else if(<var> == <int2>)  
    <statement2>;  
else  
    <statement3>;
```



# switch문 Tips

- 상수를 정의해서 사용하면 코드의 가독성이 좋아짐.
- 경우가 추가되면 상수를 새로 정의하고 case로 그 상수를 추가해주면 됨.
- fruit변수의 값을 변경할 때도 상수를 사용할 수 있음.
  - fruit = ORANGE;

```
const int APPLE = 1;
const int ORANGE = 2;
...
switch(fruit) {
    case APPLE:
        <statement1>;
        break;
    case ORANGE:
        <statement2>;
        break;
    default:
        <statement3>;
}
```

# switch문 Tips

- break문을 넣느냐 마느냐에 따라 몇 가지 경우들을 하나의 그룹으로 묶어 실행하는 것이 가능함.
- switch문은 특정 case로 점프하면 break를 만날 때까지 계속 실행한다는 점을 이용한 방법.
- 실제 case는 프로그램의 특정 영역에 레이블을 붙인 것과 같음.

```
switch(fruit) {  
    case APPLE:  
    case ORANGE:  
    case PEACH:  
        peel();  
        break;  
    case MELON:  
    case COCONUT:  
        cut();  
}
```

# Summary

- 분기문
  - if-else문
  - switch문