

Control Flow

프로그래밍 입문(2)

Topics

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

반복문이란?

- 프로그램의 특정 부분을 조건에 따라 계속 반복하여 실행하기 위한 제어문.
- 반복횟수가 정해진 경우 사용하기 좋은 for문.
- 특정 조건이 만족되는 동안 계속하여 실행할 때 좋은 while, do-while문.

for문

- ```
for (<initialization>; <termination>;
 <increment>) {
 <statement>;
}
```
- 초기화<initialization>와 종료조건<termination>을 설정하고, 매 반복(iteration)마다 <increment>를 실행하는 반복문.
- <termination>의 조건이 거짓이 되면 반복을 종료한다.
- 지정횟수만큼 반복할 때 쓰기 좋습니다.

# for문

- `int sum = 0; int size = 10;`
- `int a[size] = { 2, 4, ..., 20 };`
- `for (int i=0; i<size; i++) {  
    sum += a[i];  
}`
- 길이가 정해진 배열을 처음부터 끝까지 읽어서 더하는 코드

# Ranged for문 (C++11)

- `int sum = 0; int size = 10;`
- `int a[size] = { 2, 4, ..., 20 };`
- `for (int i : a) {  
 sum += i;  
}`
- 주어진 범위 표현(range expression)을 처음부터 끝까지 반복하며 훑는 반복문.
- 범위 표현으로는 배열 등의 순서(sequence)로 평가될 수 있는 표현식이 오면 됩니다.

# Why Ranged for?

- for문을 사용하여 무언가를 순차적으로 접근하려고 한다면, 반드시 인덱스(index)가 필요함.
- `for (int i=0; i<size; i++)`
- `i`의 값이 변하므로 `i`를 인덱스로 이용해 접근할 수 있어야 함.
- 종료 조건을 만들기 위해 반복횟수도 미리 알아내어 지정해야 함.
- Ranged for문은 특정 개체를 순차적으로 접근할 때 이런 과정이 필요없음.

# 반복문의 중첩

- for loop안에 또 다른 for loop를 쓰는 것도 가능.
- 안쪽의 for loop는 단지 바깥의 for문이 반복되는 동안 실행하는 문장의 하나로 볼 수 있음.
- ```
for(<init1>; <terminate1>; <inc1>) {  
    <statement1>;  
    for(<init2>; <terminate2>; <inc2>) {  
        <statement2>;  
    }  
    <statement3>;  
}
```


반복 횟수와 순서에 주의

- **Iteration 1:**

```
<statement1>;  
<statement2>;  
<statement2>;
```

- **Iteration 2:**

```
<statement1>;  
<statement2>;  
<statement2>;  
.....
```

- **Iteration 10:**

```
<statement1>;  
<statement2>;  
<statement2>;
```

```
for(int i=0; i<10; i++) {  
    <statement1>;  
    for(int j=0; j<2; j++) {  
        <statement2>;  
    }  
}
```

Loop with break

- switch문과 유사하게 break문을 사용하여 반복을 중단할 수 있음.
- 보통 if문과 연계하여 반복중이라도 특정 조건을 만족하면 중단하고 loop를 빠져나오도록 함.
- break는 오직 현재의 loop하나만을 빠져나온다는 점을 기억하세요.

Loop with break

- **for loop**의 Iteration 5, **for loop**의 Iteration 2에 <condition>이 만족된다고 가정.

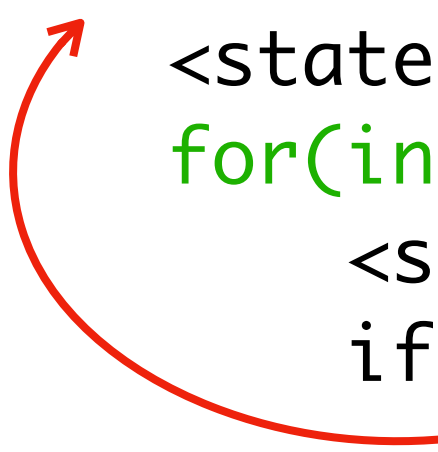
- **Iteration 5:**
<statement1>;
<statement2>;
<statement2>;

- **Iteration 6:**
<statement1>;
<statement2>;
<statement2>;
<statement2>;

- **Iteration 7: ...**

```
for(int i=0; i<10; i++) {  
    <statement1>;  
    for(int j=0; j<3; j++) {  
        <statement2>;  
        if(<condition>)  
            break;  
    }  
}
```

조건 만족시 현재 loop를 빠져나감



continue문

- break와 유사하게 반복의 흐름을 제어.
- continue문을 만나면 즉시 다음 반복으로 점프함.
- **for loop**의 Iteration 5, **for loop**의 Iteration 2에 <condition>이 만족된다고 가정.
- **Iteration 5:**
 <statement1>;
 <statement2>; **iteration 1 (j=0)**
 <statement3>;
 <statement2>; **iteration 2 (j=1)**
 <statement2>;
 <statement3>; **iteration 3 (j=2)**

```
for(int i=0; i<10; i++) {  
    <statement1>;  
    for(int j=0; j<3; j++) {  
        <statement2>;  
        if(<condition>)  
            continue;  
        <statement3>;  
    }  
}
```

조건 만족시 다음 iteration으로 진행

while문

- 조건을 만족하는 동안 계속 실행하는 반복문.
- ```
while (<condition>) {
 <statement>;
}
```
- for문과 달리 반복해서 실행하는 문장들이 while문의 조건에 관련 된 상태를 변화시켜야 함.

# while문

- `string input = " ";`
- ```
while ( !input.empty() ) {  
    getline(cin, input);  
}
```
- 들어온 입력 `input`이 비어있지 않은 동안 계속하여 실행됨.

for loop vs. while loop

- while loop는 조건을 변화시키기 위한 문장들이 복잡한 경우 사용하기 좋음.
- ```
while (!input.empty()) {
 cout << "Input your string: ";
 getline(cin, input);
}
```
- ```
for (; !input.empty(); cout << "Input your  
string: ", getline(cin, input)) {  
}
```

for loop vs. while loop

- for loop는 초기화에 선언한 변수값을 일정하게 변화시키면서 이용할 때 사용하기 좋음.
- ```
for (int i=0; i<size; i++) {
 sum += a[i];
}
```
- ```
int i = 0;  
while ( i < size ) {  
    sum += a[i];  
    i++;  
}
```


do-while문

- 우선 한 번 문장들을 실행하고, 그 이후 조건을 평가하여 만족하면 다시 반복하는 반복문.
- ```
do {
 <statement>;
} while (<condition>);
```
- while과 매우 유사하지만, while의 경우 반복을 위해서는 시작지점에는 무조건 조건을 만족시켜야만 함.
- do-while은 우선 반복할 부분을 실행하고 난 뒤 조건을 만족시키면 됨.

# do-while문

- `string input = "";` //empty string.
- ```
while ( !input.empty() ) {  
    getline(cin, input);  
}
```
- 시작시 input이 빈 문자열로 초기화 됐으므로 while안의 입력을 받는 부분은 전혀 실행되지 않음.
- ```
do {
 getline(cin, input);
} while (!input.empty());
```
- do-while문을 사용하면 우선 입력을 한 번 받고 조건을 평가할 수 있게 됨.

# Infinite Loop

- 무한히 반복되는 반복문.
- 실수로 인해 생기게 되면 프로그램이 종료되지 않음.
- 반대로 의도적으로 infinite loop를 만들고, 특정 조건에 break로 빠져나올 수 있도록 만들 수 있음.

# 의도적 Infinite Loop

- 채팅 프로그램 또는 텍스트기반 게임.
- 사용자가 명확하게 종료하겠다고 얘기하기 전까지 계속해서 입력을 받아야 함.
- ```
while (true) {  
    cout << "Input your string: ";  
    getline(cin, input);  
    if (!input.empty())  
        break;  
}
```

의도치 않은 Infinite Loop

- while문 사용시 반복하여 실행하는 문장들에 조건에 영향을 줄 수 있는 부분이 전혀 없는 경우.
- ```
while (i < size) {
 sum += a[i];
 j++;
}
```
- 또는 영원히 조건이 달성되지 않는 경우.
- ```
while ( a < b ) {  
    a++;  
    update();  
}  
void update() { b = a + 1; }
```

Summary

- 반복문
 - for문
 - while, do-while문