인트아이 정연한

2023-1 인트아이 C++ 수업 자료

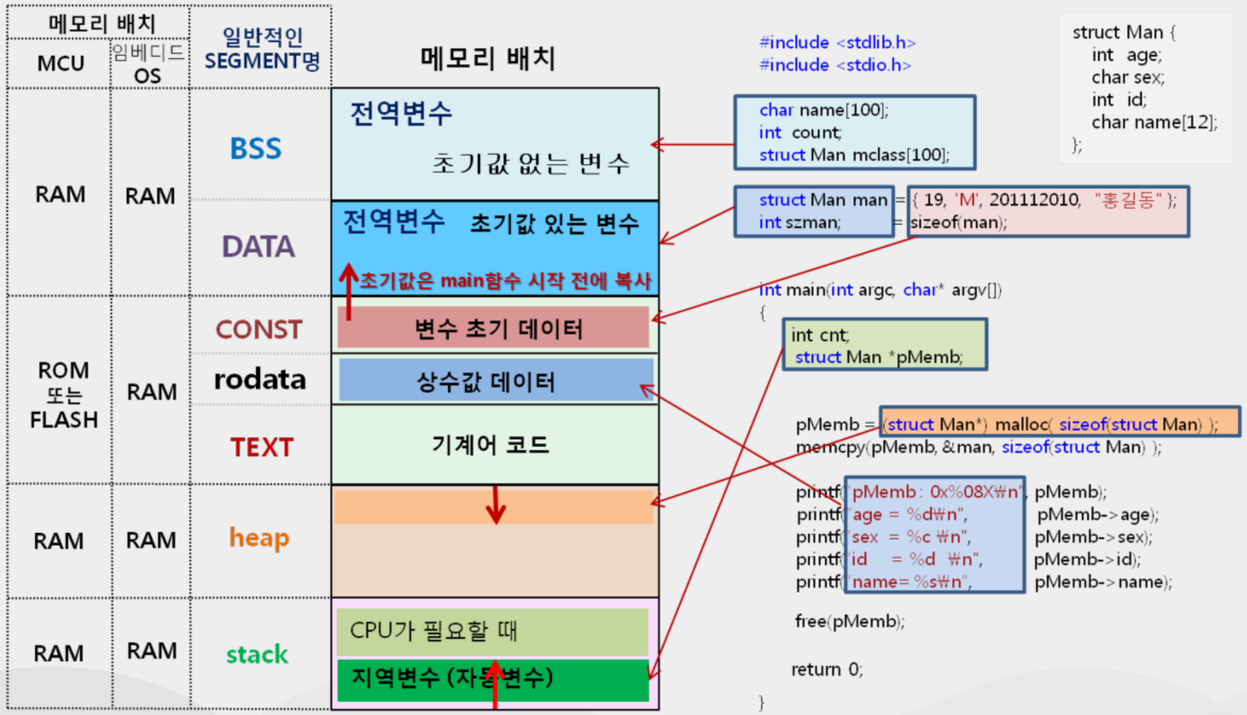
C++ 기초 2강

변수, 반복문, 조건문

CONTENTS

변수? 어디서 많이 들어봤는데

자, 변수는 간단하게는 그냥 **변할 수 있는 값**을 의미한다.
수학에서의 미지수 x와 같이 어떤 값이든 넣을 수 있는 그런 값
근데 사실 값을 넣을 수 있는 값이라는건 뭔가 좀 이상하지
프로그래밍에서의 변수는, 좀 더 엄밀히 말해서 **어떤 값을 넣기 위해 할당받은 메모리 공간**이다.
아래 그림은 메모리 구조에 관한 그림인데, 변수를 선언하면 기본적으로 이 메모리 구조의
스택 영역에서 공간을 할당받는다.



비유를 통해 이해해보자

변수 선언: 메모리(RAM)에 내가 사용할 수 있게 일정 크기의 **공간을 할당**받는 것
변수를 창고라고 치면, 땅을 임대받아 **새 창고를 짓는 행위**에 비유할 수 있음

변수의 초기화: 메모리 공간을 **할당 받음과 동시에** 그 곳에 있던 **쓰레기 값을 치우고 새 값을 넣는 것**
창고로 치면, 내가 창고를 **짓고**나서 **창고안에 있던** 건축 자재들을 **치우고**, 내가 넣고 싶은 물건을 **넣는** 행위

변수에 값 대입: **기존에 어떤 값이 들어있는** 메모리 공간에 새로운 값을 집어넣는 것
창고로 치면, 내가 **기존에 지었던** 창고에 있던 물건들을 **빼고 새 물건들을 집어넣는** 행위

자료형(data type): 데이터의 종류
기본적으로 변수를 선언하기 위해선 **변수의 이름 앞**에 반드시
해당 변수에 **들어갈** 데이터의 **자료형**을 적어주어야 한다.

변수를 선언하는 것은 창고를 짓는것에 비유할 수 있다고 했는데,
예를 들어 내가 곰인형을 창고에 넣는다고 치자.
그렇다면 창고의 형태도 **곰인형을 넣을 수 있는 창고**로 만들어야지,
뜬금없이 **냉동 창고를 만들어 버리면 안되는 것**이다.

```cpp
int main() {
    int i = 0; // 4바이트 정수 최대값: 2,147,483,647

    float f = 0.0f; // 4바이트 실수(단정밀도 부동 소수점)

    double d = 0.0; // 8바이트 실수(배정밀도 부동 소수점)

    bool b = true; // 1바이트 논리 자료형 변수에 0이 들어오면 false
    // bool 자료형 변수에 0 이외의 값이 들어오면 true
```

```cpp
// 문자는 반드시 홑따옴표로,
// 문자열은 반드시 쌍따옴표로 선언해주어야 한다.
// 문자는 말 그대로 하나의 문자(charactor), 문자열은 여러 문자들이 배열된 것
char c = 'a'; // 1바이트 문자(실제 메모리에는 해당 문자에 매칭되는 정수값으로 저장 됨)

char str1[] = "abc"; // 기존 C 스타일 문자열
string str2 = "abc"; // C++ 스타일 문자열

// 대괄호: 배열
// "abc" 대신 { 'a', 'b', 'c' } 이렇게도 넣을 수 있다.
```

```cpp
    // 변수에 값 대입
    // 자리수 구분은 홑따옴표(')를 사용해 구분할 수 있다.
    i = 2'147'483'647;

    // int 최대값에 1을 더해보자.
    // 오버플로우가 일어나 -2,147,483,648이 나오게 된다.
    cout << i + 1 << endl;

    return 0;
}
```

코드를 통해 이해해 보자

자 아래 코드를 보면 int i = 0; 이런식으로 코드가 작성되어있다.
여기서 **int**는 위에 설명 되었듯 **자료형**이다.
변수에 **들어갈 수 있는** 데이터의 **형태**를 지정해주는 역할을 하며, 여기서 **int는 정수**를 의미한다.
int의 경우 -2,147,483,648 ~ 2,147,483,647 사이의 값이 들어갈 수 있다.
그리고 자료형 옆에 있는 **i**는 **변수의 이름**을 뜻한다.
여기서 변수의 이름이 이 변수를 고유하게 식별하게 하는 **식별자**의 역할을 한다.
그리고 그 옆 = 은 변수를 **초기화** 하는 역할을 하며, 여기서는 0이란 값을 넣어 변수를 초기화 하고 있다.

```c
int main() {
    int i = 0; // 4바이트 정수 최대값: 2,147,483,647

    float f = 0.0f; // 4바이트 실수(단정밀도 부동 소수점)

    double d = 0.0; // 8바이트 실수(배정밀도 부동 소수점)

    bool b = true; // 1바이트 논리 자료형 변수에 0이 들어오면 false
    // bool 자료형 변수에 0 이외의 값이 들어오면 true
```

코드를 통해 이해해 보자

이 코드를 보면 자료형이 여러가지가 있는데, **int, float, double, bool**이 보인다.
int는 integer(정수)의 약자로, 말 그대로 **정수값**을 저장할 수 있다.
float는 floating point(부동 소수점)의 약자로 이름만 보자면 소수점이 둥둥 떠다닌다 라는 뜻인데,
실제로 float는 **실수값**을 저장하기 위해 정확도를 조금 희생하여 값에따라 소수점의 위치를 지정한다.
자세한건 https://youtu.be/ZQDsWySjY6g 참조
double는 double precision floating point(배정밀도 부동 소수점)의 약자로 float보다 좀 더 **정밀하게 실수를 표현**할 수 있다.
마지막으로 **bool**은 **논리 연산**을 위한 자료형으로 boolean algebra(불 대수)의 약자이다.
0(false)과 1(true)을 통해 논리 연산을 하기 위해 사용 되어진다.

```
int main() {
    int i = 0; // 4바이트 정수 최대값: 2,147,483,647

    float f = 0.0f; // 4바이트 실수(단정밀도 부동 소수점)

    double d = 0.0; // 8바이트 실수(배정밀도 부동 소수점)

    bool b = true; // 1바이트 논리 자료형 변수에 0이 들어오면 false
    // bool 자료형 변수에 0 이외의 값이 들어오면 true
```

문자는 어떻게 표현하지?

위에서 말한 4가지 자료형 외에도 자주 쓰이는 자료형이 하나 더 있는데, 바로 **char**이다.
char은 **하나의 문자**를 저장할 수 있는 자료형으로, 기본적으로 **알파벳과 몇몇 특수문자**를 저장할 수 있다.
아니 그럼 한글은 어떻게 저장하냐? 할 수 있는데 **한글은 문자열로 저장**할 수 있다.
문자열이란 문자들이 모인 것으로, 한글 한 글자는 **1바이트로 표현 할 수 없으며**, UTF-8 인코딩 기준 3바이트를 차지한다.
visual studio는 기본적으로 EUC-KR 인코딩을 사용하여 한글을 저장하는데, 이 인코딩 기준 한글은 2바이트를 차지한다.
UTF-8이니, EUC-KR이니 뭔소린지 모르겠다면 일단 한글은 문자열을 사용해 저장한다는 것만 알아두면 된다.
인코딩 관련한 설명은 나중에 번외편으로 따로 올릴 예정이다.

```
// 문자는 반드시 홑따옴표로,
// 문자열은 반드시 쌍따옴표로 선언해주어야 한다.
// 문자는 말 그대로 하나의 문자(charactor), 문자열은 여러 문자들이 배열된 것
char c = 'a'; // 1바이트 문자(실제 메모리에는 해당 문자에 매칭되는 정수값으로 저장 됨)

char str1[] = "abc"; // 기존 C 스타일 문자열
string str2 = "abc"; // C++ 스타일 문자열

// 대괄호: 배열
// "abc" 대신 { 'a', 'b', 'c' } 이렇게도 넣을 수 있다.
```

그렇다면 문자열은?

아무튼 그래서 문자는 char을 이용해 저장 할 수 있다는 건 알겠는데, 문자열은 어떻게 표현하느냐,
바로, **char 배열** 또는 C++의 문자열을 저장하는, 자료구조의 일종인 **string**을 통해 저장할 수 있다.
우선 char 배열부터 보자, 코드를 보면 char str1[] = "abc"라고 되어있는데,
딱 보아하니 문자열은 쌍따옴표를 통해서 선언할 수 있고, 이렇게 선언된 문자열을 저 str1[]라는 놈한테 넣는 것 같다.

그런데 여길 보면 처음 보는 기호(**[]**)가 하나 등장하는데, 이 기호는 배열을 선언하는 **선언자**로,
변수를 선언 할 때, 이 배열 선언자를 붙이면 일반적인 변수가 아닌 **배열을 선언**하게 된다.

```
// 문자는 반드시 홑따옴표로,
// 문자열은 반드시 쌍따옴표로 선언해주어야 한다.
// 문자는 말 그대로 하나의 문자(charactor), 문자열은 여러 문자들이 배열된 것
char c = 'a'; // 1바이트 문자(실제 메모리에는 해당 문자에 매칭되는 정수값으로 저장 됨)

char str1[] = "abc"; // 기존 C 스타일 문자열
string str2 = "abc"; // C++ 스타일 문자열

// 대괄호: 배열
// "abc" 대신 { 'a', 'b', 'c' } 이렇게도 넣을 수 있다.
```

그렇다면 문자열은?

이때, 문자열은 문자들이 **정해진 순서대로 나열**된 것, 즉 **배열 된 문자들**을 뜻한다.
따라서 문자열을 저장하기 위해서, 문자 배열(char array)을 사용할 수 있는데,
여기선 char str1[]을 통해 문자 배열을 선언하였다.
이렇게 선언된 문자 배열에 "abc"를 넣게 되면, str1의 크기가 정해지는데,
이때, **문자열의 끝**에는 반드시 **₩0**이라는 **제어 문자**가 들어가게 된다.
이 문자는 문자열의 끝을 나타내는 문자로, 이 문자로 인해 str1은 **abc₩0 이렇게 4개의 문자가 저장**되게 된다.
이 때문에 str1의 **크기**는 이 초기화 시점에 **4**로 지정되게 된다.(문자 하나가 1바이트이므로)

```
// 문자는 반드시 홑따옴표로,
// 문자열은 반드시 쌍따옴표로 선언해주어야 한다.
// 문자는 말 그대로 하나의 문자(charactor), 문자열은 여러 문자들이 배열된 것
char c = 'a'; // 1바이트 문자(실제 메모리에는 해당 문자에 매칭되는 정수값으로 저장 됨)

char str1[] = "abc"; // 기존 C 스타일 문자열
string str2 = "abc"; // C++ 스타일 문자열

// 대괄호: 배열
// "abc" 대신 { 'a', 'b', 'c' } 이렇게도 넣을 수 있다.
```

그렇다면 문자열은?

문자 배열은 대충 알아봤으니, **string**에 대해 살펴보자.
string은 다른 자료형들과는 다르게, 하이라이트가 파란색이 아닌 초록색으로 들어가있다.
이는 다른 자료형들과 달리 class로 선언 된 **자료구조**이기 때문인데,
string은 **C++에서 기본 지원**하는 자료 구조 중 하나로, **문자를 요소로 가지는 자료구조**이다.
따라서 문자 배열처럼, 문자열을 집어넣을 수 있는 **컨테이너**인데,
내부에 여러가지 **메서드**를 가지고 있어, 각종 **연산**이 쉽게 가능하다는 장점이 있다.
또한 보기에도 훨씬 **직관적**이기 때문에 앞으로 **문자열을 사용할때는 string을 사용**해주도록 하자.

```
// 문자는 반드시 홑따옴표로,
// 문자열은 반드시 쌍따옴표로 선언해주어야 한다.
// 문자는 말 그대로 하나의 문자(charactor), 문자열은 여러 문자들이 배열된 것
char c = 'a'; // 1바이트 문자(실제 메모리에는 해당 문자에 매칭되는 정수값으로 저장 됨)

char str1[] = "abc"; // 기존 C 스타일 문자열
string str2 = "abc"; // C++ 스타일 문자열

// 대괄호: 배열
// "abc" 대신 { 'a', 'b', 'c' } 이렇게도 넣을 수 있다.
```

대입 그리고 오버플로우

아래 코드를 보면 아까 초기화 할 때 처럼 변수 i에 **= 연산자를 사용**해 값을 집어넣고 있다.
하지만 이건 **초기화가 아니라 대입**이다.
변수가 **선언될 때 값을 넣어주는 것**만이 **초기화**이고 **그 이외의 상황에 값을 넣어주는 것**은 전부 **대입**이다.
자 그런데 여기서 int 최대값을 i에 넣어주었는데, 여기에 1을 더한 값을 출력하게 되면,
오버플로우가 일어나 int 최소값인 -2,147,483,648이 나오게 된다.

```
    // 변수에 값 대입
    // 자리수 구분은 홑따옴표(')를 사용해 구분할 수 있다.
    i = 2'147'483'647;

    // int 최대값에 1을 더해보자.
    // 오버플로우가 일어나 -2,147,483,648이 나오게 된다.
    cout << i + 1 << endl;

    return 0;
}
```

대입 그리고 오버플로우

오버플로우가 뭔지, 오버플로우가 나면 왜 음수가 나오는지에 대한 **자세한 설명은 GitHub에 올린 1-2 코드**에 자세히 올려두었다.
여기서는 오버플로우가 뭔지에 대해서만 간략하게 설명하도록 하겠다.
기본적으로 오버플로우란 **나타낼 수 있는 값의 상한 또는 하한을 넘어서 값이 증가/감소** 하게 되면,
해당 자료형이 나타낼 수 있는 **가장 작은 값 또는 가장 큰 값**을 나타내는 것을 말한다.
아래 예시에선 3비트를 예시로 들었는데, 최상위 비트를 부호비트로 사용하였기 때문에 나타낼 수 있는 값의 범위가 -4~3이다.
예시를 잘 보면 011 까지는 양수 3이었는데 거기에 1을 증가시켜보니 100 이 되면서, 부호비트를 침범해, 음수인 -4가 되었다.

```
// 3비트로 줄여서 보면
// 3비트로 나타낼 수 있는 값의 범위 = -4~3
// 2^3 == 8개
// 0을 양수 취급하기 때문에 양수의 범위는
// 해당 변수가 나타낼 수 있는 최대값 / 2 - 1 이 되는 것이고
// 음수는 그런 거 없으니까 그냥 최대값 / 2가 되는 것이고
// 000 == 0
// 001 == 1
// 010 == 2
// 011 == 3
// 100 == -4
// 101 == -3
// 110 == -2
// 111 == -1
// 000 == 0
```

같은 작업을 여러번 시도하려면

기본적으로 같은 작업을 여러번 시도한다 했을 때, 직관적으로는 단순히 해당 작업을 하는 코드를
여러번 작성하면 된다고 생각할 것이다.
그러나 C++에는 이런 귀찮은 작업을 대신 해주는 문법이 있는데, 이름하야 반복문이다.
이름 그대로 반복을 위해 사용되는 문법인데, 기본적으로 **for, while, do-whlie**이 있다.
아래는 반복문을 왜 써야 하는지에 대한 예시이다.

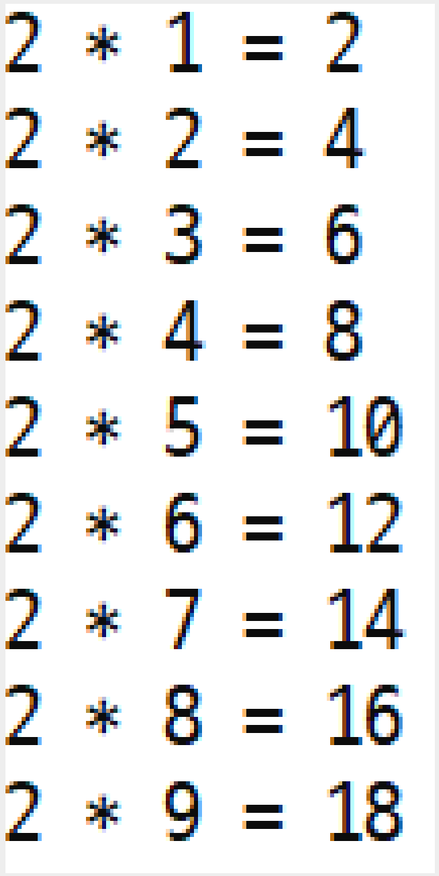
```
// 만약 반복문이 없다면...
void 구구단(int 단) {
    cout << 단 << " * 1 = " << 1 * 단 << endl;
    cout << 단 << " * 2 = " << 2 * 단 << endl;
    cout << 단 << " * 3 = " << 3 * 단 << endl;
    cout << 단 << " * 4 = " << 4 * 단 << endl;
    cout << 단 << " * 5 = " << 5 * 단 << endl;
    cout << 단 << " * 6 = " << 6 * 단 << endl;
    cout << 단 << " * 7 = " << 7 * 단 << endl;
    cout << 단 << " * 8 = " << 8 * 단 << endl;
    cout << 단 << " * 9 = " << 9 * 단 << endl;
}
```

for문으로 구구단을 만들어보자

여러 반복문 중, for문은 순차 반복에 용이하다.
아래처럼 코드를 작성하고, 실행을 시켜보면 오른쪽 사진처럼 구구단 2단이 출력되게 된다.

코드를 분석해보면, for문은 for(선언부; 조건부; 증감부) 로 나뉘게 되는데,
선언부: 반복을 위한 변수를 선언하는 부분
조건부: 선언부에서 선언한 변수를 통해 반복을 할지 말지를 결정하는 조건을 체크한다.
증감부: for문의 내용을 다 실행한 후, 반복을 위한 변수를 증가 또는 감소 시키는 역할을 한다.

```cpp
int main() {
    // i나 j같은 반복을 위한 변수를 통해 반복 조건을 체크한다.
    for (int i = 1; i < 10; ++i) {
        cout << 2 << " * " << i << " = " << 2 * i << endl;
    }
    return 0;
}
```



코드를 분석해보자

코드를 보면 for (int i = 1; i < 10; ++i) 라고 되어있는데,
우선 i 를 선언하고 1로 초기화 후, 조건을 검사한다.
이때 i는 1이기 때문에 i < 10 이라는 조건에 부합하고 따라서 for 문 안의 내용을 실행시킨다.
그리고 for 문 안의 내용을 다 실행시켰다면, 다시 맨 위로 올라와 ++i를 통해 i 의 값을 1 증가시키고,
다시 조건을 검사해준다.
이때, i는 2로, 여전히 10 미만이므로, 다시 한 번 반복하고, 이것을 9번 반복해준다.

정리하자면, i 선언 -> 조건 검사 -> 본문 실행 -> i값 증가 -> 조건 검사 -> 본문 실행 -> ... -> 조건 검사 -> 탈출

```cpp
int main() {
    // i나 j같은 반복을 위한 변수를 반복자라 한다.
    for (int i = 1; i < 10; ++i) {
        cout << 2 << " * " << i << " = " << 2 * i << endl;
    }
    return 0;
}
```

```
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
```

while문은 뭐지?

while 문은 while (조건부) 의 형태로 이루어져있으며, 보면 알겠지만 for문에서 선언부와 증감부가 빠진 형태이다.
while 문 역시 조건부의 내용이 true 일 때 반복문의 본문을 실행시키는데,
for 문이 순차 반복이라면, while 문은 반복조건이 좀 더 강조 된, 조건 반복이라 볼 수 있다.
아래의 코드를 보면 반복을 위한 변수가 선언되지도 않았는데도 잘만 반복이 되는 것을 볼 수 있다.

```cpp
// while (조건부)

int main() {
    int a = 0;
    cout << "10을 입력해주세요" << endl;
    cin >> a;

    // a가 10이 아닐 때, 반복
    while (a != 10) {
        cout << "a가 10이 아닙니다. 다시 입력해주세요." << endl;
        cin >> a;
    }
    return 0;
}
```

```
10을 입력해주세요
4
a가 10이 아닙니다. 다시 입력해주세요.
8
a가 10이 아닙니다. 다시 입력해주세요.
6
a가 10이 아닙니다. 다시 입력해주세요.
10
```

do-while문은 또 뭐지?

do-while문은 while문과 비슷하지만, 조건을 검사하지 않고, 최소 한 번은 실행하는 반복문으로
반복문의 본문에서 반복 조건에 해당하는 값이 변경되는 경우 등 아주 제한적인 경우에 사용 된다.
오른쪽의 코드를 보면, 문자열을 입력받고, 그 문자열 안의 각 문자를 소문자 -> 대문자, 대문자 -> 소문자
로 바꾸는 동작을 하고 있는데, while 조건을 보니 문자 != '₩0' 가 조건으로 들어와 있는 것을 볼 수 있다.

이것은 문자열 안의 특정 문자가 ₩0 이라는 문자가 아닐 때 반복을 하겠다 라는 뜻인데,
₩0는 제어문자라 불리는 특수한 문자 중 하나인 NULL문자로 ₩와 0이 각각 따로가 아니라
₩0 자체가 하나의 문자이다.
여기서는 폰트의 차이로 역슬래시가 아니라 원화기호로 표시되지만, 실제로는 해당 기호는 역슬래시가 맞다.

아무튼 ₩0은 문자열의 끝을 나타내는 제어 문자이기 때문에, 저 조건은 다시말해
문자열의 끝에 도달하면 반복을 종료하겠다. 라는 뜻이 된다.

```
int main() {
    string 문자열 = "";
    char 문자 = '\0'; // 문자를 입력 받을 변수 선언
    int i = 0; // 반복을 위한 반복자

    cout << "문자열 입력 : ";
    //cin >> 문자열; // 띄어쓰기는 포함 안함
    getline(cin, 문자열); // 띄어쓰기를 포함 함

    do {
        문자 = 문자열[i];

        if (문자 >= 'A' && 문자 <= 'Z')
            문자열[i] = 소문자로_변환(문자);
        else if (문자 >= 'a' && 문자 <= 'z')
            문자열[i] = 대문자로_변환(문자);
        else
            문자열[i] = 문자;
        i++;
    } while (문자 != '\0'); // NULL문자

    cout << "변환된 결과 ==> " << 문자열 << endl;
}
```

```
문자열 입력 : abCDeFg
변환된 결과 ==> ABcdEfG
```

조건분기를 만들어내는 조건문

C++에서 조건문은 두가지가 있는데, 하나는 if문이고, 나머지 하나는 switch-case문이다.
조건문은 조건분기를 만들어내는데,
조건 분기란, 특정한 조건에 부합하는 경우, 이하에 속하는 지시사항들의 실행 여부를 결정하는 일을 의미한다.
예를 들어, 내가 차를 타고 가고 있다고 가정해보자. 이때, 신호등이 있다면 신호등의 신호에 따라 행동을 달리할 것이다.

아래의 코드를 보면 신호등의 값에 따라 차가 갈지 말지를 결정하는 조건 분기가 있다.
신호등이 빨간불이라면 멈출 것이고, 빨간불이 아니라 노란불이라면 속도를 줄일 것이고,
빨간불도, 노란불도 아니라면 원래 속도로 지나갈 것이다.

```cpp
int main() {
    int 신호등 = 빨간불;

    if (신호등 == 빨간불) {
        cout << "차를 멈췄습니다." << endl;
    }
    else if (신호등 == 노란불) {
        cout << "속도를 줄여 천천히 지나갑니다." << endl;
    }
    else {
        cout << "원래 속도로 지나갑니다." << endl;
    }
    return 0;
}
```

```
차를 멈췄습니다.
```

switch-case문은 뭐지?

이번에도 신호등을 예시로 들었다.
switch문은 특이하게 조건식에 오는 값이 어떤 값인지에 따라서 케이스가 나뉘게되는데,
신호등의 값이 빨간불이라면, case 빨간불에 멈출 것이고,
신호등의 값이 노란불이라면, case 노란불에 멈출 것이고,
신호등의 값이 빨간불도 노란불도 아니라면 default에 멈출것이다.
추가로, case는 정수값만 올 수 있는데, 지금 보기에는 정수가 아니라 웬 변수가 들어가 있는 것 처럼 보인다.
이것은 enum을 쓴 것인데, 이것은 추후 class를 나가면서 같이 배울 예정이다.

```
int main() {
    int 신호등 = 노란불;

    switch (신호등) {
    case 빨간불:
        cout << "차를 멈췄습니다." << endl;
        break;
    case 노란불:
        cout << "속도를 줄여 천천히 지나갑니다." << endl;
        break;
    default:
        cout << "원래 속도로 지나갑니다." << endl;
        break;
    }

    return 0;
}
```

```
속도를 줄여 천천히 지나갑니다.
```

백준 10871번 X보다 작은 수

문제

정수 N개로 이루어진 수열 A와 정수 X가 주어진다. 이때, A에서 X보다 작은 수를 모두 출력하는 프로그램을 작성하시오.

입력

첫째 줄에 N과 X가 주어진다. (1 ≤ N, X ≤ 10,000)
둘째 줄에 수열 A를 이루는 정수 N개가 주어진다. 주어지는 정수는 모두 1보다 크거나 같고, 10,000보다 작거나 같은 정수이다.

출력

X보다 작은 수를 입력받은 순서대로 공백으로 구분해 출력한다. X보다 작은 수는 적어도 하나 존재한다.

예제 입력 1 복사	예제 출력 1 복사
10 5 1 10 4 9 2 3 8 5 7 6	1 4 2 3