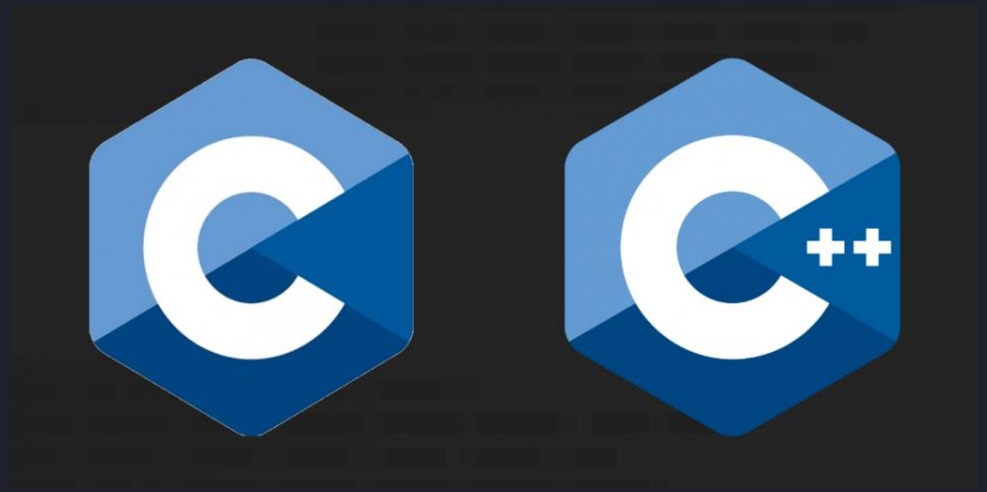

SPLIT 튜터링 2회차

:array와 반복문 기초

기계항공공학부 17학번 김기성

Contents

namespace: ‘std::’의 의미

- 여러 회사들이 협업을 하는 경우를 상상해봅시다.
  - A회사에서 만든 코드와 B회사에서 만든 코드를 모두 쓰려고 하는데, hello라는 같은 이름을 가진 함수가 있습니다.
  - 이 두 파일을 include한 후 hello라는 함수를 호출하면 어떤 hello함수가 사용되는 걸까요?
- 코드에도 소속이 있어야 합니다.
  - 즉, 서울 사는 김상덕씨와 알래스카에 사는 김상덕씨를 구별할 방법이 필요합니다.
  - 이를 해결하기 위해 namespace라는 개념을 사용합니다.

```
#include "foo.h"//A회사에서 작성한 코드
#include "goo.h"//B회사에서 작성한 코드

int main()
{
    std::cout << hello() << std::endl;
    return 0;
}
```

namespace: ‘std::’의 의미

- 협업을 할 때, 이름 충돌을 방지하기 위해 namespace 안에 자신의 코드를 작성합니다.
- 특정 네임스페이스의 함수나 변수 등을 사용하기 위해 ‘::’ 연산자를 사용합니다.
- 그렇다면 std::cout, std::cin, std::endl을 생각해보면,
  - 아마 이것들이 std라는 namespace에 정의되어있구나라고 생각할 수 있겠죠?
  - C++에서 기본적으로 제공하는 standard library는 모두 std namespace에 포함되어 있습니다.

foo.h:

```
namespace Foo {
    int hello() {
        return 3;
    }
}
```

goo.h:

```
namespace Goo {
    int hello() {
        return 5;
    }
}
```

```
#include "foo.h"
#include "goo.h"

int main() {
    std::cout << Foo::hello() << std::endl;
    std::cout << Goo::hello() << std::endl;
}
```

using namespace: 여긴 알래스카야!

- std::cout, std::cin, std::endl,,, 매번 std라는 namespace를 지정해주기가 번거롭습니다.
- using namespace std;를 선언해줌으로써 이 파일에서는 std라는 namespace를 기본적으로 사용한다는 것을 명시적으로 선언해줍니다.
- 여긴 알래스카라고 선언하면, '김상덕씨'는 기본적으로 '알래스카 김상덕씨'이고 '서울 사는 김상덕씨'를 따로 불러주면 되겠죠.

```
#include <iostream>

using namespace std;

int main() {
    cout << "Hello, World!" << endl;
    std::cout << "Hello, World!" << std::endl;
}
```

복잡한 변수 다루기-array

- C++에서는 변수의 array를 선언할 수 있습니다.
- 선언 방법은 “type array_name[array_size]”입니다.
- array에 접근하는 방법은 “array_name[index]”입니다.
  - 참고: 0-based indexing입니다. 길이 N의 array의 인덱스는 0, 1, … , N-1까지입니다.

```
#include <iostream>

using namespace std;

int main() {
    int my_array[3];
    my_array[0] = 1;
    my_array[1] = 10;
    my_array[2] = 100;
    cout << my_array[0] << endl;
    cout << my_array[1] << endl;
    cout << my_array[2] << endl;
    return 0;
}
```

복잡한 변수 다루기-array

- array를 좀 더 간단하게 초기화하는 방법
  - 중괄호 안에 원소들을 직접 입력해 '=' 연산자를 통해 초기화합니다.
- 중괄호를 통해 초기화할 경우 array_size를 생략할 수 있습니다.
- 입력한 array_size보다 중괄호를 통해 입력한 원소가 적을 경우, 뒤쪽은 기본값으로 초기화됩니다.
  - 기본값은 int, double의 경우 0, bool의 경우 false입니다.

```
#include <iostream>

using namespace std;

int main() {
    int array1[5] = { 1, 2, 3, 4, 5 };
    int array2[] = { 1, 2, 3, 4, 5 }; //{1, 2, 3, 4, 5}
    int array3[5] = { 1, 2, 3 }; //{1, 2, 3, 0, 0}
    int array4[5] = { 0 }; //{0, 0, 0, 0, 0}
    return 0;
}
```

복잡한 변수 다루기-문자열

- char 자료형은 한 글자만 저장할 수 있습니다.
- 그런데 긴 문자열은 어떻게 변수로 저장할 수 있을까요?
  - char의 array를 이용하면 됩니다.
- 일반 array와 다른 점은 array의 size를 (내가 입력하고자 하는 길이+1) 이상으로 잡아야 합니다.
  - 그 이유는, 문자열 끝을 표시하기 위한 종료문자('₩0')의 자리가 필요하기 때문입니다.
  - 필요이상으로 잡는 건 상관 없습니다.
- 문자열 전체를 출력하려면 어떻게 해야할까요?

```
#include <iostream>

using namespace std;

int main() {
    char a[6] = "hello";
    return 0;
}
```

복잡한 변수 다루기-문자열


```cpp
#include <iostream>

using namespace std;

int main() {
    char a[5];
    a[0] = 'h';
    a[1] = 'e';
    a[2] = 'l';
    a[3] = 'l';
    a[4] = 'o';
    for (int i = 0; i < 5; ++i){
        cout << a[i];
    }
    cout << endl;
    return 0;
}
```

종료문자 없는 세상에서 프로그래머가 해야하는 일

```cpp
#include <iostream>

using namespace std;

int main() {
    char a[6] = { ‘h’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’ };
    cout << a; // '\0' 이전까지 있는 모든 문자를 출력함
    return 0;
}
```

종료문자가 있어 간단해진 입출력

반복문의 기초: while문

- 반복문은 특정 조건을 만족하는 동안 body에 해당하는 코드가 실행됩니다.
- while(조건){ /* 코드 블록 */ }
- 예시) 다음과 같은 출력을 하는 코드를 작성하여라.
  - 1번째 루프입니다.
  - 2번째 루프입니다.
  - 3번째 루프입니다.
  - 4번째 루프입니다.
  - 5번째 루프입니다.

```cpp
#include <iostream>

using namespace std;

int main() {
    int count = 1;
    while (count <= 5) {
        cout << count << "번째 루프입니다." << endl;
        count += 1;
    }
    return 0;
}
```

반복문의 기초: while문

- while문을 이용해 다음과 같이 응용할 수 있습니다.
- 문제) 다음과 같이 동작하는 프로그램을 작성해봅시다.

```
N을 입력하세요: 5
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
```

반복문의 기초: while문

- 솔루션)

```cpp
#include <iostream>

using namespace std;

int main() {
    cout << "N을 입력하세요: ";
    int N;
    cin >> N;
    int i = 0;
    while (i < N) {
        cout << "Hello, World!" << endl;
        i += 1;
    }
    return 0;
}
```

반복문의 기초: while문

- while문을 사용했을 때의 문제:
  - 무한루프에 빠지기 쉽습니다.
  - 무한루프란, body를 아무리 실행해도 종료 조건에 다다르지 않아 끝나지 않는 상태를 말합니다.
  - 사실은... while문의 잘못이 아니라 프로그래머의 잘못이긴 합니다...
  - body를 실행함에 따라 종료조건에 가까워지는지 꼭! 확인해야 합니다.
- 다음 시간에는 이런 human error를 줄일 수 있는 for문에 대해서 알아보도록 하겠습니다.

```
#include <iostream>

using namespace std;
int main() {
    cout << "N을 입력하세요: ";
    int N;
    cin >> N;
    int i = 0;
    while (i < N) {
        cout << "Hello, World!" << endl;
        }
    return 0;
}
```

반복문의 기초: while문

- 여담: while문의 body가 한 줄이라면, 중괄호를 쓸 필요가 없습니다.
  - C++에서 하나의 statement(문장)은 ';'(세미콜론) 하나 혹은, '{}'중괄호로 감싸진 여러 문장입니다.
  - while문에서는 while문 이후에 있는 딱 하나의 statement를 body로 생각합니다.
  - 그럼 아래 두 코드를 실행하면 어떤 어떤 차이가 있을까요? 실행해서 확인해봅시다.

```cpp
#include <iostream>

using namespace std;

int main() {
    cout << "N을 입력하세요: ";
    int N;
    cin >> N;
    int i = 0;
    while (i < N)
        i += 1;
    cout << "i 는 " << i << "입니다." << endl;
    return 0;
}
```

```cpp
#include <iostream>

using namespace std;

int main() {
    cout << "N을 입력하세요: ";
    int N;
    cin >> N;
    int i = 0;
    while (i < N);
        i += 1;
    cout << "i 는 " << i << "입니다." << endl;
    return 0;
}
```

연습문제

- 연습문제) 아래와 같은 동작을 하는 프로그램을 작성해봅시다.
  - sum이라는 변수를 0으로, i라는 정수를 1로 초기화 한다.
  - 입력할 정수의 개수 N을 선언하고 입력받는다.
  - i가 N보다 같거나 작은 동안 “i번째 정수를 입력하세요:”를 출력하고 한 정수(temp)를 입력받는다.
  - 입력받은 정수를 sum에 더해준다.
  - i에 1을 더해준다.
  - while문을 빠져나왔다면, “N개 정수의 합은 sum입니다.”를 출력한다.

```
입력할 정수의 개수를 입력하세요: 5
1번째 정수를 입력하세요: 1
2번째 정수를 입력하세요: 2
3번째 정수를 입력하세요: 3
4번째 정수를 입력하세요: 4
5번째 정수를 입력하세요: 5
5개 정수의 합은 15입니다.
```

연습문제

- 연습문제2) 문자열의 길이를 출력하는 프로그램을 작성해봅시다. 단, 입력의 길이는 100을 넘지 않습니다.
  - 힌트: 문자열의 종료문자가 '₩0'임을 이용해봅시다.

```
문자열을 입력하세요: hello
문자열의 길이는 5입니다.
```