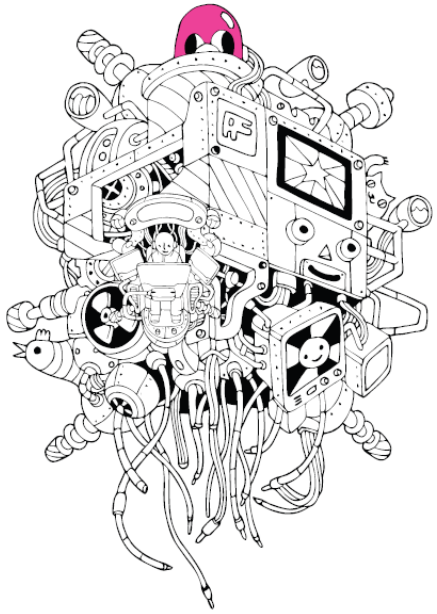
윤성우의
열혈 C++ 프로그래밍



윤성우 저 열혈강의 C++ 프로그래밍 개정판

Chapter 01. C언어 기반의 C++ 1

윤성우의

열혈 C++ 프로그래밍

Chapter 01-1. printf와 scanf를 대신하는 입출력 방식

윤성우 저 열혈강의 C++ 프로그래밍 개정판

C++ 버전의 Hello World 출력 프로그램

예제를 통해서 확인할 사실 몇 가지

- 헤더파일의 선언 — #include <iostream>
- 출력의 기본구성 — std::cout<<'출력대상1'<<'출력대상2'<<'출력대상3';
- 개행의 진행 — std::endl을 출력하면 개행이 이뤄진다.

```
#include <iostream>

int main(void)
{
    int num=20;
    std::cout<<"Hello World!"<<std::endl;
    std::cout<<"Hello "<<"World!"<<std::endl;
    std::cout<<num<<' '<<'A';
    std::cout<<' '<<3.14<<std::endl;
    return 0;
}
```

C언어에서는 출력의 대상에 따라 서식지정을 달리했지만, C++에서는 그러한 과정이 불필요하다.

실행결과

```
Hello World!
Hello World!
20 A 3.14
```

예제를 이해하려 들지 말고 관찰하자. 그리고 위의 세 가지 사실을 확인하자.

scanf를 대신하는 데이터의 입력

예제를 통해서 확인할 사실 몇 가지

- 입력의 기본구성 — std::cin>>'변수'
- 변수의 선언위치 — 함수의 중간 부분에서도 변수의 선언이 가능하다.

```
#include <iostream>

int main(void)
{
    int val1;
    std::cout<<"첫 번째 숫자입력: ";
    std::cin>>val1;
    int val2;
    std::cout<<"두 번째 숫자입력: ";
    std::cin>>val2;
    int result=val1+val2;
    std::cout<<"덧셈결과: "<<result<<std::endl;
    return 0;
}
```

C++에서는 변수의 선언위치에 제한을 두지 않는다.

출력에서와 마찬가지로 입력에서도 별도의 서식 지정이 불필요하다.

실행결과

```
첫 번째 숫자입력: 3
두 번째 숫자입력: 5
덧셈결과: 8
```

C++의 지역변수 선언


```
#include <iostream>

int main(void)
{
    int val1, val2;
    int result=0;
    std::cout<<"두 개의 숫자입력: ";
    std::cin>>val1>>val2;
    if(val1<val2)
    {
        for(int i=val1+1; i<val2; i++)
            result+=i;
    }
    else
    {
        for(int i=val2+1; i<val1; i++)
            result+=i;
    }
    std::cout<<"두 수 사이의 정수 합: "<<result<<std::endl;
    return 0;
}
```

이렇듯 연이은 데이터의 입력을 명령할 수 있다.

for문 안에서도 변수의 선언이 가능하다.

std::cin을 통해서 입력되는 데이터의 구분은 스페이스 바, 엔터, 탭과 같은 공백을 통해서 이뤄진다.

실행결과

```
두 개의 숫자입력: 3 7
두 수 사이의 정수 합: 15
```

배열 기반의 문자열 입출력


```
#include <iostream>

int main(void)
{
    char name[100];
    char lang[200];
    std::cout<<"이름은 무엇입니까? ";
    std::cin>>name;
    std::cout<<"좋아하는 프로그래밍 언어는 무엇인가요? ";
    std::cin>>lang;

    std::cout<<"내 이름은 "<<name<<"입니다.\n";
    std::cout<<"제일 좋아하는 언어는 "<<lang<<"입니다."<<std::endl;
    return 0;
}
```

문자열의 입력방식도 다른 데이터의 입력방식과 큰 차이가 나지 않는다.

```
이름은 무엇입니까? Yoon
좋아하는 프로그래밍 언어는 무엇인가요? C++
내 이름은 Yoon입니다.
제일 좋아하는 언어는 C++입니다.
```

실행결과

윤성우의

열혈 C++ 프로그래밍

Chapter 01-2. 함수 오버로딩 (Function Overloading)

윤성우 저 열혈강의 C++ 프로그래밍 개정판

함수 오버로딩의 이해


```
int MyFunc(int num)
{
    num++;
    return num;
}
```

```
int MyFunc(int a, int b)
{
    return a+b;
}
```

```
int main(void)
{
    MyFunc(20);    // MyFunc(int num) 함수의 호출
    MyFunc(30, 40);    // MyFunc(int a, int b) 함수의 호출
    return 0;
}
```

C++은 함수호출 시 '함수의 이름'과 '전달되는 인자의 정보'를 동시에 참조하여 호출할 함수를 결정한다. 따라서 이렇듯 매개변수의 선언이 다르다면 동일한 이름의 함수도 정의 가능하다. 그리고 이러한 형태의 함수 정의를 가리켜 '함수 오버로딩(Function Overloading)'이라 한다.

함수 오버로딩의 예


```
int MyFunc(char c) { . . . }
int MyFunc(int n) { . . . }
```

매개변수의 자료형이 다르므로 함수 오버로딩 성립

```
int MyFunc(int n) { . . . }
int MyFunc(int n1, int n2) { . . . }
```

매개변수의 수가 다르므로 함수 오버로딩 성립

```
void MyFunc(int n) { . . . }
int MyFunc(int n) { . . . }
```

반환형의 차이는 함수 오버로딩의 조건을 만족시키지 않는다.

윤성우의

열혈 C++ 프로그래밍

Chapter 01-3. 매개변수의 디폴트 값

윤성우 저 열혈강의 C++ 프로그래밍 개정판

매개변수에 설정하는 '디폴트 값'의 의미


```
int MyFuncOne(int num=7)
{
    return num+1;
}
```

인자를 전달하지 않으면 7이 전달된 것으로 간주한다. 여기서의 디폴트 값은 7! 따라서 이 함수를 대상으로 하는 다음 두 함수의 호출은 그 결과가 같다.

```
MyFuncOne( );
MyFuncOne(7);
```

```
int MyFuncTwo(int num1=5, int num2=7)
{
    return num1+num2;
}
```

인자를 전달하지 않으면 각각 5와 7이 전달된 것으로 간주한다. 따라서 이 함수를 대상으로 하는 다음 두 함수의 호출은 그 결과가 같다.

```
MyFuncTwo( );
MyFuncTwo(5, 7);
```

디폴트 값은 함수의 선언에만 위치


```
#include <iostream>
int Adder(int num1=1, int num2=2);

int main(void)
{
    std::cout<<Adder()<<std::endl;
    std::cout<<Adder(5)<<std::endl;
    std::cout<<Adder(3, 5)<<std::endl;
    return 0;
}

int Adder(int num1, int num2)
{
    return num1+num2;
}
```

함수의 선언을 별도로 둘 때에는 디폴트 값의 선언을 함수의 선언부에 위치시켜야 한다. 그 이유는 컴파일러의 컴파일 특성에서 찾을 수 있다.

컴파일러는 함수의 디폴트 값의 지정여부를 알아야 함수의 호출문장을 적절히 컴파일 할 수 있다.

부분적 디폴트 값 설정


```
int YourFunc(int num1, int num2=5, int num3=7) { . . . }

 YourFunc(10);          // YourFunc(10, 5, 7);
 YourFunc(10, 20);      // YourFunc(10, 20, 7);
```

매개변수의 일부에만 디폴트 값을 지정하고, 채워지지 않은 매개변수에만 인자를 전달하는 것이 가능하다.

```
int YourFunc(int num1, int num2, int num3=30) { . . . }          (○)
int YourFunc(int num1, int num2=20, int num3=30) { . . . }       (○)
int YourFunc(int num1=10, int num2=20, int num3=30) { . . . }    (○)
```

전달되는 인자가 왼쪽에서부터 채워지므로, 디폴트 값은 오른쪽에서부터 채워져야 한다.

```
int WrongFunc(int num1=10, int num2, int num3) { . . . }         (×)
int WrongFunc(int num1=10, int num2=20, int num3) { . . . }      (×)
```

전달되는 인자가 왼쪽에서부터 채워지므로, 오른쪽이 빈 상태로 왼쪽의 매개변수에만 일부 채워진 디폴트 값은 의미를 갖지 못한다. 따라서 컴파일 에러를 일으킨다.

윤성우의

열혈 C++ 프로그래밍

Chapter 01-4. 인라인(inline) 함수

윤성우 저 열혈강의 C++ 프로그래밍 개정판

매크로 함수의 장점과 함수의 inline 선언


```
#define SQUARE(x) ((x)*(x))

int main(void)
{
    std::cout<< SQUARE(5) <<std::endl;
    return 0;
}
```

선행처리 결과

```
int main(void)
{
    std::cout<< ((5)*(5)) <<std::endl;
    return 0;
}
```

장점. 함수가 인라인화 되어 성능의 향상으로 이어질 수 있다.

단점. 함수의 정의 방식이 일반함수에 비해서 복잡하다. 따라서 복잡한 함수의 정의에는 한계가 있다.

```
inline int SQUARE(int x)
{
    return x*x;
}

int main(void)
{
    std::cout<<SQUARE(5)<<std::endl;
    std::cout<<SQUARE(12)<<std::endl;
    return 0;
}
```

키워드 inline 선언은 컴파일러에 의해서 처리된다. 따라서 컴파일러가 함수의 인라인화를 결정한다.

inline 선언이 되어도 인라인처리 되지 않을 수 있고, inline 선언이 없어도 인라인처리 될 수 있다.

실행결과

```
25
144
```

매크로 함수의 장점은 취하고 단점은 보완한 것이 C++의 인라인 함수이다.

인라인 함수에는 없는 매크로 함수만의 장점


```
#define SQUARE(x) ((x)*(x))
```

매크로 함수는 자료형에 독립적이다.

```
std::cout<< SQUARE(12);        // int형 함수호출
std::cout<< SQUARE(3.15);      // double형 함수호출
```

inline 선언된 함수를 위의 형태로 호출하려면, 각 자료형 별로 함수가 오버로딩 되어야 한다. 즉, 매크로 함수와 달리 자료형에 독립적이지 못하다.

```
template <typename T>
inline T SQUARE(T x)
{
    return x*x;
}
```

inline 함수를 자료형에 독립적으로 선언하는 방법! 이는 이후에 템플릿을 통해서 학습하게 된다.

윤성우의

열혈 C++ 프로그래밍

Chapter 01-5. 이름공간에 대한 소개

윤성우 저 열혈강의 C++ 프로그래밍 개정판

이름공간의 기본원리

존재하는 이름공간이 다르면 동일한 이름의 함수 및 변수를 선언하는 것이 가능하다.

프로젝트의 진행에 있어서 발생할 수 있는 이름의 충돌을 막을 목적으로 존재하는 것이 이름공간이다.

```
namespace BestComImpl
{                                   BestComImpl이라는 이름의 공간
    void SimpleFunc(void)
    {
        std::cout<<"BestCom이 정의한 함수"<<std::endl;
    }
}

namespace ProgComImpl
{                                   ProgComImpl이라는 이름의 공간
    void SimpleFunc(void)
    {
        std::cout<<"ProgCom이 정의한 함수"<<std::endl;
    }
}

int main(void)
{                       범위 지정 연산자
    BestComImpl::SimpleFunc();
    ProgComImpl::SimpleFunc();
    return 0;
}
```

이름공간 BestComImpl에 정의된 SimpleFunc의 호출

이름공간 ProgComImpl에 정의된 SimpleFunc의 호출

이름공간 기반의 함수 선언과 정의의 분리


```
namespace BestComImpl
{
    void SimpleFunc(void);
}
namespace ProgComImpl
{
    void SimpleFunc(void);
}
int main(void)
{
    BestComImpl::SimpleFunc();
    ProgComImpl::SimpleFunc();
    return 0;
}
void BestComImpl::SimpleFunc(void)
{
    std::cout<<"BestCom이 정의한 함수"<<std::endl;
}
void ProgComImpl::SimpleFunc(void)
{
    std::cout<<"ProgCom이 정의한 함수"<<std::endl;
}
```

이름공간 BestComImpl에 위치하는 함수 SimpleFunc의 선언과 정의의 분리

이름공간 ProgComImpl에 위치하는 함수 SimpleFunc의 선언과 정의의 분리

동일한 이름공간 내에서의 함수호출

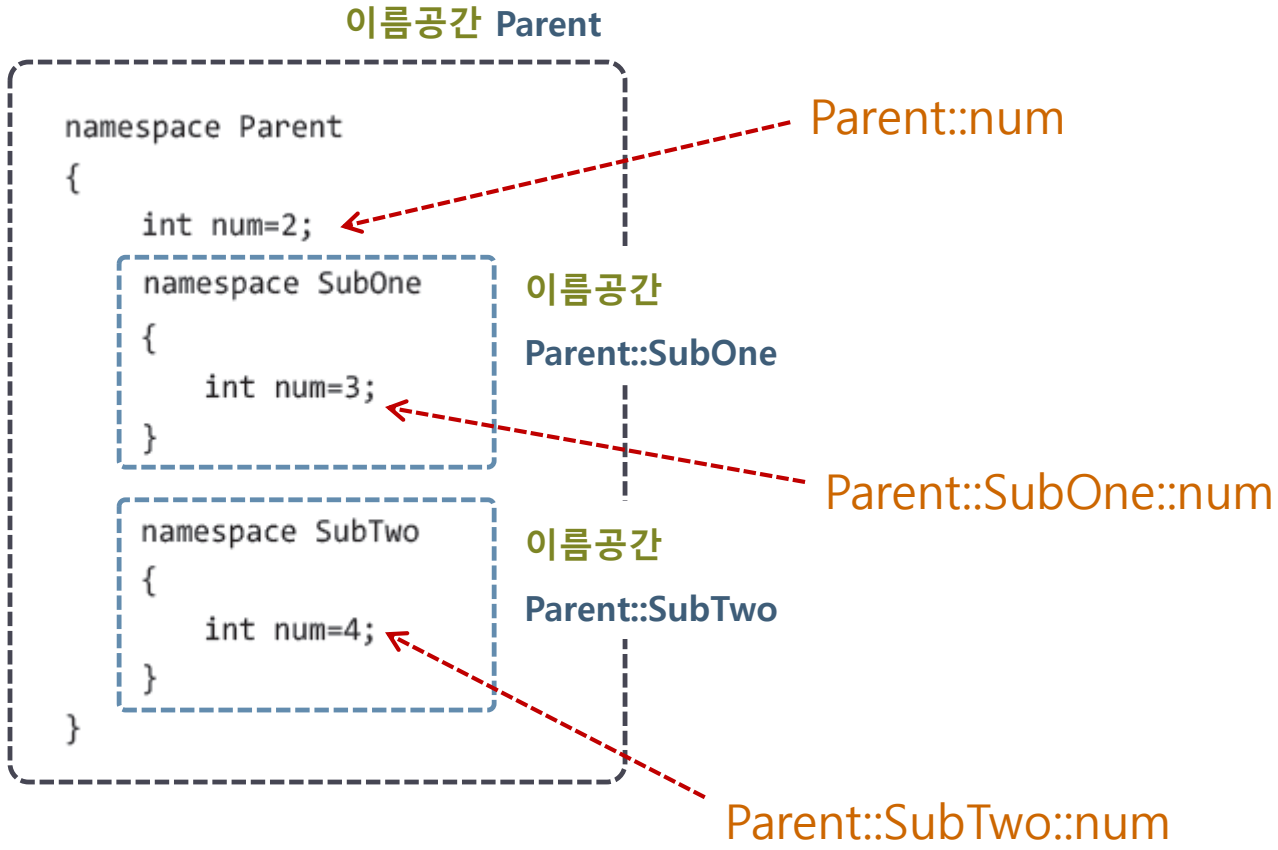

```
namespace BestComImpl
{
    void SimpleFunc(void);
}
namespace BestComImpl
{
    void PrettyFunc(void);
}
```

선언된 이름공간의 이름이 동일하다면, 이 둘은 동일한 이름공간으로 간주한다. 즉, SimpleFunc와 PrettyFunc는 동일한 이름 공간안에 존재하는 상황이다.

```
void BestComImpl::SimpleFunc(void)
{
    std::cout<<"BestCom이 정의한 함수"<<std::endl;
    PrettyFunc();                // 동일 이름공간
    ProgComImpl::SimpleFunc();   // 다른 이름공간
}
void BestComImpl::PrettyFunc(void)
{
    std::cout<<"So Pretty!!"<<std::endl;
}
```

이름공간을 명시하지 않고 함수를 호출하면, 함수의 호출문이 존재하는 함수와 동일한 이름공간 안에서 호출할 함수를 찾게 된다. 따라서 SimpleFunc 함수 내에서는 이름공간을 명시하지 않은 상태에서 PrettyFunc 함수를 직접호출 할 수 있다.

이름공간의 중첩



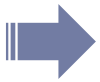
이름공간은 중첩이 가능하다. 따라서 계층적 구조를 갖게끔
이름공간을 구성할 수 있다.

std::cout, std::cin, std::endl

std::cout → "이름공간 std에 선언된 cout"

std::cin → "이름공간 std에 선언된 cin"

std::endl → "이름공간 std에 선언된 endl"



```
namespace std
{
    cout . . . .
    cin . . . .
    endl . . . .
}
```

<iostream>에 선언되어 있는 cout, cin 그리고 endl은 이름공간 std 안에 선언되어 있다.

이렇듯 이름충돌을 막기 위해서, C++ 표준에서 제공하는 다양한 요소들은 이름공간 std 안에 선언되어 있다.

using을 이용한 이름공간의 명시


```
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

int main(void)
{
    int num=20;
    cout<<"Hello World!"<<endl;
    cout<<"Hello "<<"World!"<<endl;
    cout<<num<<' '<<'A';
    cout<<' '<<3.14<<endl;
    return 0;
}
```

이후부터 cin, cout, endl은

std::cin, std::cout, std::endl을 의미한다는 선언

```
#include <iostream>
using namespace std;

int main(void)
{
    int num=20;
    cout<<"Hello World!"<<endl;
    cout<<"Hello "<<"World!"<<endl;
    cout<<num<<' '<<'A';
    cout<<' '<<3.14<<endl;
    return 0;
}
```

이름공간 std에 선언된 것은 std라는 이름공간의

선언없이 접근하겠다는 선언

너무 빈번한 using namespace의 선언은 이름의 충돌을 막기위한 이름공간의 선언을 의미 없게 만든다. 따라서 제한적으로 사용할 필요가 있다.

이름공간의 별칭 지정과 전역변수의 접근


```
namespace AAA
{
    namespace BBB
    {
        namespace CCC
        {
            int num1;
            int num2;
        }
    }
}
```

```
namespace ABC=AAA::BBB::CCC;
```

AAA::BBB::CCC에 대해 ABC라는 이름의 별칭 선언 후,

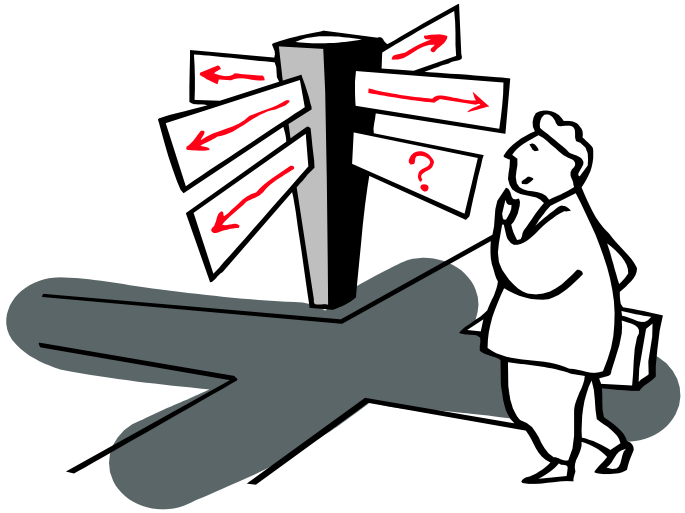
```
ABC::num1=10;
ABC::num2=20;
```

위와 같이 하나의 별칭으로 이름공간의 선언을 대신할 수 있다.

```
int val=100;        // 전역변수

int SimpleFunc(void)
{
    int val=20;     // 지역변수
    val+=3;         // 지역변수 val의 값 3 증가
    ::val+=7;       // 전역변수 val의 값 7 증가
}
```

범위지정 연산자는 지역변수가 아닌 전역변수의 접근에도 사용이 가능하다.



Chapter 01이 끝났습니다. 질문 있으신지요?