

# C++ Programming

4<sup>th</sup> Study: From C to C++ (4/4)

- Reference
- Function overloading
- Namespace

C++ Korea 옥찬호 (utilForever@gmail.com)



# Reference

An alias to an already-existing object or function

# Reference

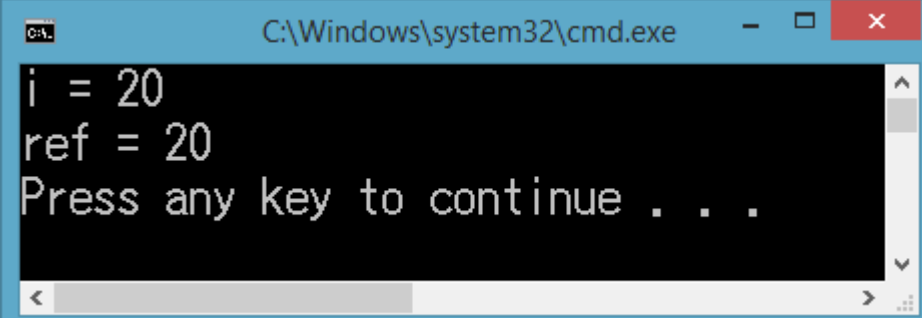
- 레퍼런스는 다른 변수를 가리키는 변수 (포인터 기능의 일부)
  - 다른 변수와 동일한 메모리 위치를 서로 공유함
  - 레퍼런스에는 포인터나 가리키는 값을 사용하거나 레퍼런스에 값을 대입할 때 \*나 &를 추가해야 하는 특별한 문법이 필요없음
  - 한마디로 포인터의 간소화 버전! (변수에 별명을 붙인다고 생각하면...)
- 레퍼런스는 언제나 유용한 메모리만을 가리켜야 함
  - 레퍼런스를 선언하는 방법 : `int& ref;`
  - 하지만 레퍼런스는 반드시 초기화해야 함! 위와 같이 선언하면 안 됨  
→ `int x = 5; int& ref = x;`

# Reference: Example

```
int main()
{
    // int& ref;
    int i = 10;
    int& ref = i;

    ref += 10;
    std::cout << "i = " << i
               << std::endl;
    std::cout << "ref = " << ref
               << std::endl;

    return 0;
}
```



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background with white text. The output displayed is:

```
i = 20
ref = 20
Press any key to continue . . .
```

The window includes standard Windows window controls (minimize, maximize, close) in the top right corner and a scroll bar on the right side.

# Reference: Example

## Pass-by-pointer

```
void swap(int* a, int* b)
{
    int temp = *a; *a = *b; *b = temp;
}

int main()
{
    int a = 10, b = 20;

    std::cout << "a = " << a << ", b = " << b << std::endl;
    swap(&a, &b);
    std::cout << "a = " << a << ", b = " << b << std::endl;

    return 0;
}
```

# Reference: Example

## Pass-by-reference

```
void swap(int& a, int& b)
{
    int temp = a; a = b; b = temp;
}

int main()
{
    int a = 10, b = 20;

    std::cout << "a = " << a << ", b = " << b << std::endl;
    swap(a, b);
    std::cout << "a = " << a << ", b = " << b << std::endl;

    return 0;
}
```

# Function overloading

The ability to create multiple methods of the same name

# Function overloading

- 사각형의 넓이를 구하는 방법...?
  - 사각형의 네 점으로 면적을 계산하는 방법
    - `int computeRectArea(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);`
  - 사각형의 너비와 높이로 계산하는 방법
    - `int computeRectArea(int width, int height);`라고 하고 싶은데,  
똑같은 이름을 써서 두 함수가 충돌할 것 같고, 그렇다고 떠오르는 이름도 없고...
- C와 달리, C++에서는 함수 오버로딩(Overloading)이라는 개념을 지원하기 때문에 걱정할 일이 없음
  - 오버로딩이란, 여러 함수에서 같은 이름을 사용할 수 있다는 뜻



# Function overloading

- 다만, 함수 오버로딩에는 몇 가지 규칙이 있음
  - 오버로딩된 함수는 인수의 개수가 서로 달라야 함
    - `int sum(int num1, int num2);`  
`int sum(int num1, int num2, int num3);`
  - 인수의 개수가 같은 경우에는 인수의 타입이 달라야 함
    - `void print(int num1, int num2);`  
`void print(double num1, double num2);`
  - 리턴 타입만 다른 경우에는 같은 이름의 함수를 사용할 수 없음
    - `void print(int num1, int num2);`  
`int print(int num1, int num2);`  
→ 컴파일 오류!

# Function overloading: Example

C

```
int computeRectArea1(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4)
{
    return 0;
}

int computeRectArea2(int width, int height) {
    return 0;
}

int main()
{
    int area1 = computeRectArea1(1, 1, 1, 4, 3, 4, 3, 1);
    int area2 = computeRectArea2(2, 3);

    return 0;
}
```

# Function overloading: Example

C++

```
int computeRectArea(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4)
{
    return 0;
}

int computeRectArea(int width, int height) {
    return 0;
}

int main()
{
    int area1 = computeRectArea(1, 1, 1, 4, 3, 4, 3, 1);
    int area2 = computeRectArea(2, 3);

    return 0;
}
```

# Namespace

A method for preventing name conflicts in large projects

# Namespace

- 이러한 상황을 가정해 보면...
  - 어떤 프로젝트를 진행하는데, 두 사람에게 각자 일을 맡김
  - 이후, 구현한 코드에서 변수 및 함수명이 서로 똑같은 사실을 발견함
  - 충돌을 피하기 위해, 한 사람의 코드를 바꾸고 싶지만  
그러기에는 바꿔야 하는 양이 너무 많음 → 문제 발생!
- 문제를 해결하기 위해서는, 타입의 기본 이름을 확장해야 함
  - C에서는 이러한 문제를 해결할 방법이 없었음
  - C++에서는 네임스페이스(Namespace)를 사용해 이 문제를 해결

# Namespace

- 네임스페이스를 쉽게 이해하려면...
  - 한 아파트에 철수라는 이름을 가진 사람이 10명 있다고 가정
  - 아파트 밖에서 “철수야~”라고 부르면 10명 전부 대답 → 충돌!
  - 하지만 “101동 302호에 사는 철수야~”라고 부르면 1명만 대답  
→ 네임스페이스를 사용하면 충돌을 피할 수 있음!

# Namespace

- 네임스페이스를 사용하는 방법 : namespace 이름 { ... }
  - 중첩해서 사용할 수도 있음
    - namespace com { namespace cpp { ... } }
- 네임스페이스 A에 있는 변수 a에 접근하고 싶다면, A::a
  - 여기서 ::는 범위 지정 연산자: A에 있는 a
  - std::cout도 마찬가지! : 네임스페이스 std에 있는 cout
- 네임스페이스를 매번 적기 귀찮다면, using을 사용
  - 예 : using namespace A; 또는 using namespace std;
    - A::a → a, std::cout → cout

# Namespace: Example

C

```
void print()
{
    std::cout << "A's print" << std::endl;
}

void print()
{
    std::cout << "B's print" << std::endl;
}

int main()
{
    print();

    return 0;
}
```

	Code	Description ▾
✖	C2084	function 'void print(void)' already has a body
✖	C3861	'print': identifier not found



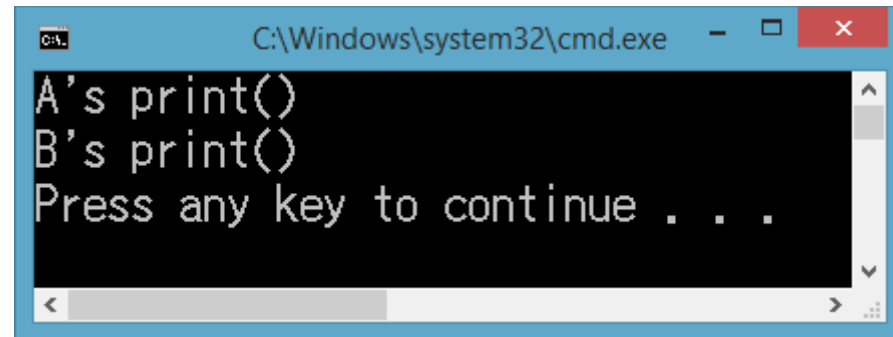
# Namespace: Example

C++

```
namespace A {  
    void print() { std::cout << "A's print()" << std::endl; }  
}  
  
namespace B {  
    void print() { std::cout << "B's print()" << std::endl; }  
}  
  
int main()  
{  
    A::print();  
    B::print();  
  
    return 0;  
}
```

# Namespace: Example

C++



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background with white text. It displays the following output:  
A's print()  
B's print()  
Press any key to continue . . .  
The window includes standard Windows window controls (minimize, maximize, close) in the title bar and a scroll bar on the right side.

# Namespace: Example

C++

```
using namespace std;

namespace A {
    void print() { cout << "A's print()" << endl; }
}

namespace B {
    void print() { cout << "B's print()" << endl; }
}

int main()
{
    A::print();
    B::print();
}
```

# Namespace: Example




```
namespace A { int i; }
namespace B { int i; }

using namespace A;
using namespace B;

int main()
{
    // Don't do this
    if (i == i) { }

    if (A::i == B::i) { }

    return 0;
}
```

	Code	Description ▾
		IntelliSense: "i" is ambiguous
		IntelliSense: "i" is ambiguous
	C2872	'i': ambiguous symbol