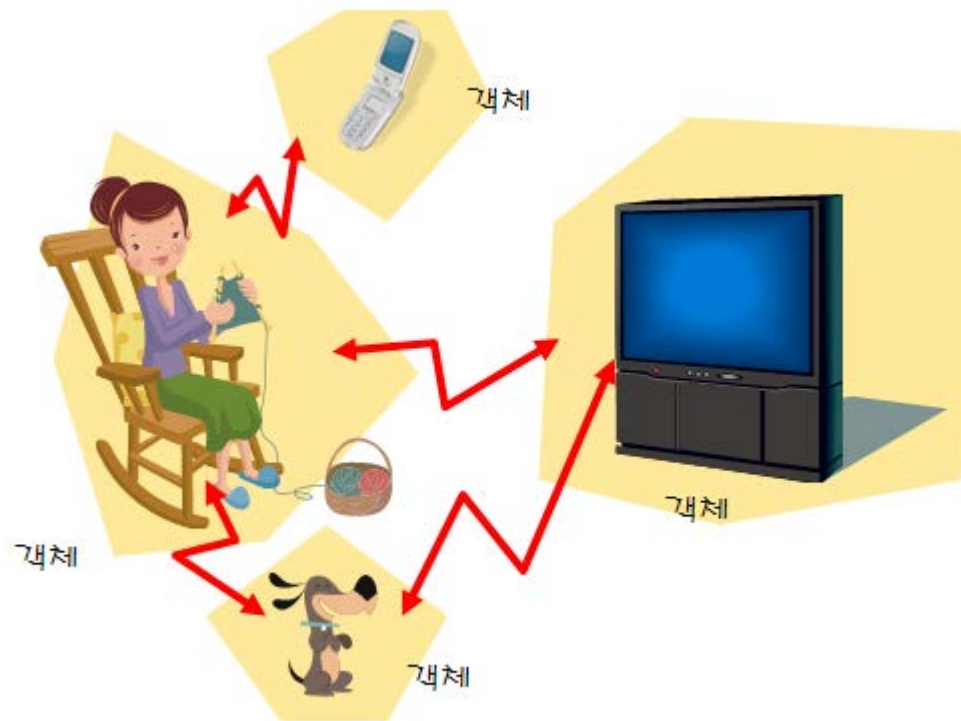




Power C++

제7장 C언어의 향상





이번 장에서 학습할 내용



- 레퍼런스
- 디폴트 매개 변수
- 인라인 함수
- 동적 메모리 할당
- 이름 공간

이번 장에서는
C언어를
향상시킨
부분에 대하여
학습한다.





변수 선언 위치 변경

```
int getSum(int array[], int n)
{
    cout << "변수는 어디서나 선언이 가능합니다" << endl;
    int sum=0;
    for(int i=0 ; i < n; i++)
        sum += array[i];
    return 0;
}
```



형변환 방법의 변경

```
double d = 3.14;
```

```
int i;
```

```
i = (int)d;
```

```
i = int(d); // 새로운 형변환 형식
```



엄격한 자료형 검사

```
sub(int n)           // int sub(int n)으로 수정하여야 함
{
    ...
    return 0;
}
```



bool형

- 참과 거짓을 나타내기 위한 bool형을 도입

```
bool isLarger=false;  
  
if( x > y )  
    isLarger = true;  
else  
    isLarger = false;
```



구조체

- 구조체는 변수들을 모아 놓은 것(8장에서 학습)
- 태그만 가지고서도 구조체 변수 정의 가능

```
struct _point {  
    int x;  
    int y;  
};  
struct _point p1; // C 언어 방식  
_point p2;      // C++ 언어 방식
```



중간 점검 문제

1. `bool` 타입의 변수가 가질 수 있는 값은?
2. C++에서 함수의 중간에서 변수가 선언될 수 있는가?





레퍼런스

- 레퍼런스(reference): 변수에 별명을 붙이는 것

`int &ref = var;`

- “레퍼런스 ref은 변수 var의 별명(alias)이다”





예제

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

int main()
{
    int var;
    int &ref = var;           // 레퍼런스 선언

    var = 10;
    cout << "var의 값: " << var << endl;
    cout << "ref의 값: " << ref << endl;

    ref = 20;                 // ref의 값을 변경하면 var의 값도 변경된다.
    cout << "var의 값: " << var << endl;
    cout << "ref의 값: " << ref << endl;

    return 0;
}
```



```
var의 값: 10
ref의 값: 10
var의 값: 20
ref의 값: 20
```



주의할 점

- `int n=10, m=20;`
- `int &ref = n;`
- `ref = m;` // 컴파일 오류!!
- `int &ref;` // 컴파일 오류!!
- `int &ref = 10;` // 컴파일 오류!!

변경 불가

초기화되지 않았음

상수로 초기화할 수 없음



참조에 의한 호출(레퍼런스)

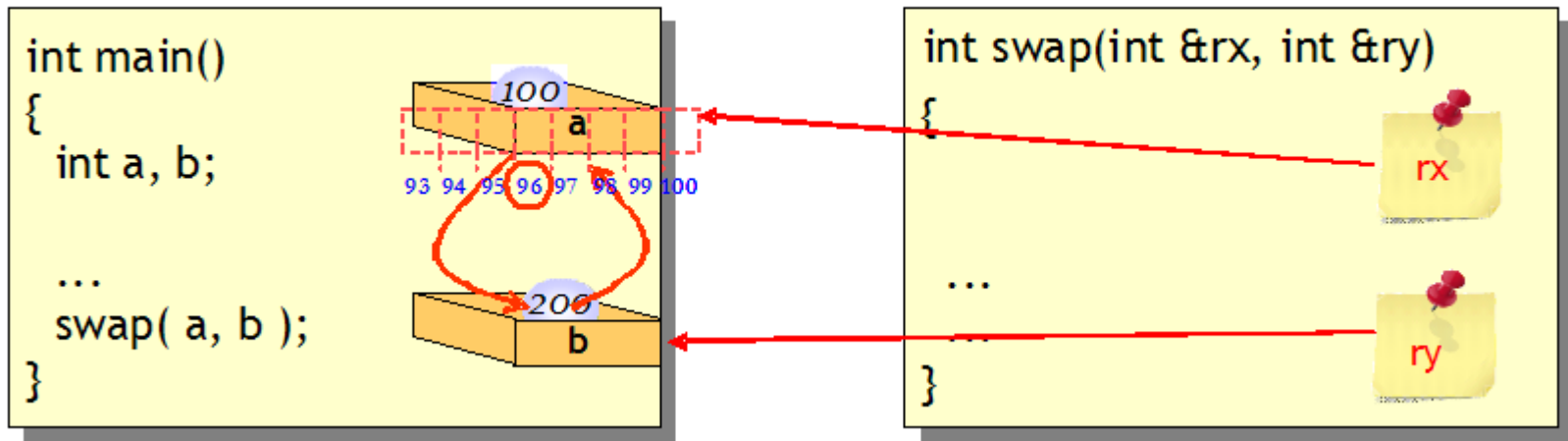
```
#include <iostream>
using namespace std;
void swap(int &rx, int &ry);
int main()
{
    int a = 100, b = 200;
    cout << "swap() 호출전: a = " << a << ", b = " << b << endl;
    swap(a, b);
    cout << "swap() 호출후: a = " << a << ", b = " << b << endl;
    return 0;
}
void swap(int &rx, int &ry)
{
    int tmp;
    cout << "In swap() : rx = " << rx << ", ry = " << ry << endl;
    tmp = rx; rx = ry; ry = tmp;
    cout << "In swap() : rx = " << rx << ", ry = " << ry << endl;
}
```



```
swap() 호출전: a = 100, b = 200
In swap() : rx = 100, ry = 200
In swap() : rx = 200, ry = 100
swap() 호출후: a = 200, b = 100
```



참조에 의한 호출(레퍼런스)





레퍼런스를 이용한 반환



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

enum RCODE { SUCCESS, ERROR };

// 기울기와 y절편을 계산
RCODE get_line_parameter(int x1, int y1, int x2, int y2, float &slope, float &yintercept)
{
    if( x1 == x2 )
        return ERROR;
    else
    {
        slope = (float)(y2 - y1)/(float)(x2 - x1);
        yintercept = y1 - slope*x1;
        return SUCCESS;
    }
}
```



레퍼런스를 이용한 반환

```
int main()
{
    float s, y;

    if( get_line_parameter(3, 3, 6, 6, s, y) == ERROR )
        cout << "에러" << endl;
    else
        cout << "기울기는 " << s << endl << "y절편은 " << y << endl;

    return 0;
}
```

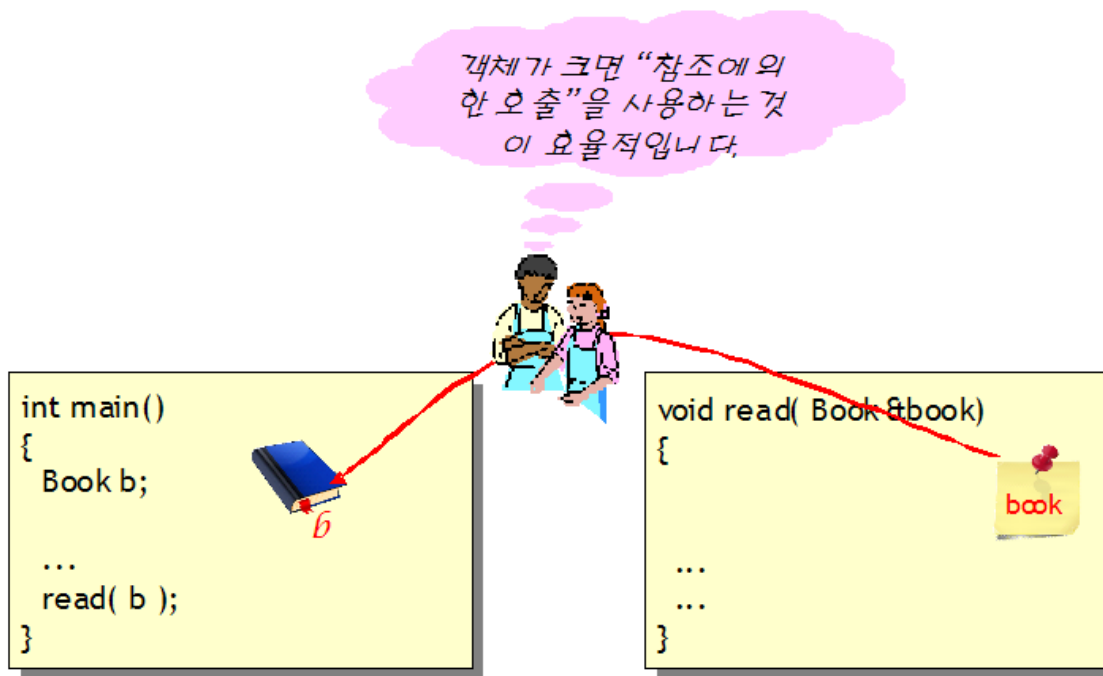


기울기는 1
y절편은 0
계속하려면 아무 키나 누르십시오 . . .



레퍼런스를 통한 효율성 향상

- 객체의 크기가 큰 경우, 복사는 시간이 많이 걸린다. 이때는 레퍼런스로 처리하는 것이 유리





레퍼런스를 통한 변경을 방지하려면

- `const`를 앞에 붙이면 레퍼런스가 가리키는 내용이 변경 불가능한 상수라는 뜻

```
void sub(const int &p)
{
    p = 0; // 오류!!
}
```



포인터 vs 레퍼런스

- 일반적으로 레퍼런스를 사용하는 편이 쉽다.
- 만약 참조하는 대상이 수시로 변경되는 경우에는 포인터를 사용
- NULL이 될 가능성이 있는 경우에도 포인터를 사용

```
int *p = new int;  
if( p != NULL )  
{  
    int &ref = *p;  
    ref = 100;  
}
```



중간 점검 문제

1. 포인터와 레퍼런스의 차이점을 설명하라.
2. 레퍼런스보다 포인터를 사용하여야 하는 경우는?
3. 함수가 레퍼런스를 반환할 수 있는가?

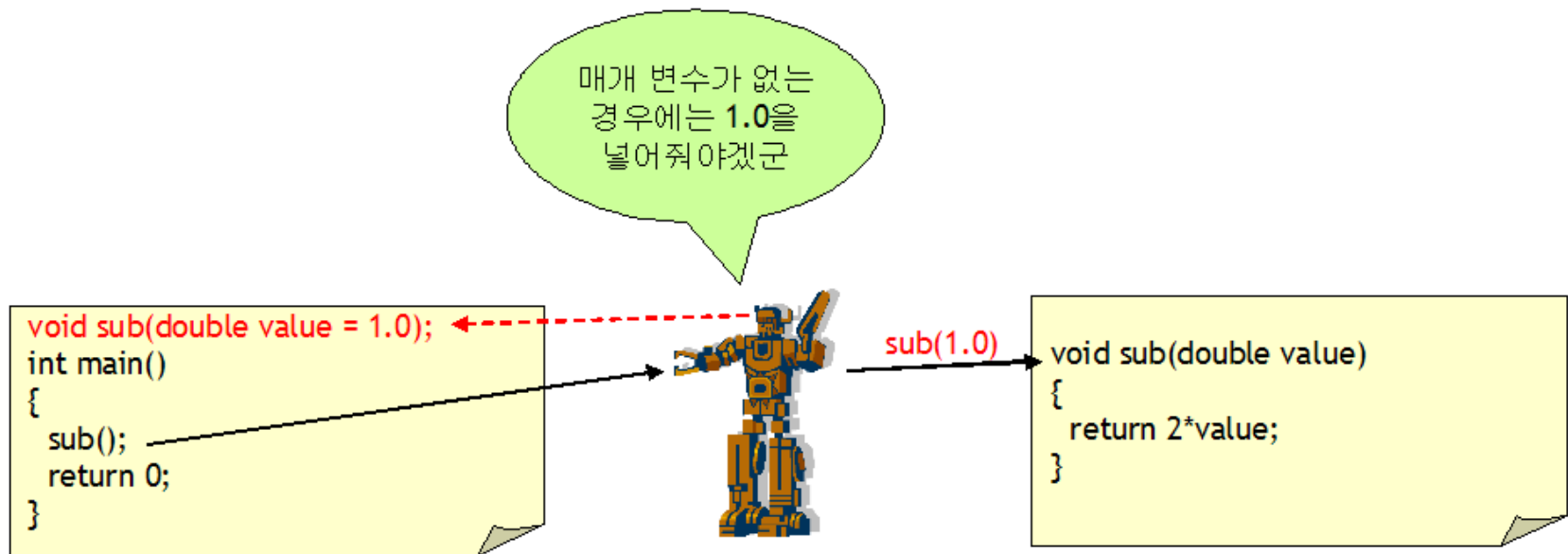




디폴트 매개 변수

- 디폴트 매개 변수(default parameter): 인자를 전달하지 않아도 디폴트값을 대신 넣어주는 기능

```
void sub(double value = 1.0);    // 함수 원형 정의시
```





주의할 점

- 디폴트 매개 변수는 뒤에서부터 앞쪽으로만 정의할 수 있다.

```
void sub(int p1, int p2, int p3=30);// OK!
```

```
void sub(int p1, int p2=20, int p3=30);// OK!
```

```
void sub(int p1=10, int p2=20, int p3=30);// OK!
```

```
void sub(int p1, int p2=20, int p3);// 컴파일 오류!
```

```
void sub(int p1=10, int p2, int p3=30);// 컴파일 오류!
```



예제



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

```
int calc_deposit(int salary=300, int month=12);
```

```
int main()
```

```
{
```

```
    cout << "0개의 디폴트 매개 변수 사용" << endl;
```

```
    cout << calc_deposit(200, 6) << endl;
```

```
    cout << "1개의 디폴트 매개 변수 사용" << endl;
```

```
    cout << calc_deposit(200) << endl;
```

```
    cout << "2개의 디폴트 매개 변수 사용" << endl;
```

```
    cout << calc_deposit() << endl;
```

```
    return 0;
```

```
}
```

```
int calc_deposit(int salary, int month)
```

```
{
```

```
    return salary*month;
```

```
}
```



0개의 디폴트 매개 변수 사용

1200

1개의 디폴트 매개 변수 사용

2400

2개의 디폴트 매개 변수 사용

3600

계속하려면 아무 키나 누르십시오 . . .



중복 함수

- 중복 함수(overloading functions):
같은 이름을 가지는 함수를 여러 개 정의하는 것

// 정수값을 제공하는 함수

```
int square(int i)
```

```
{
```

```
    return i*i;
```

```
}
```

// 실수값을 제공하는 함수

```
double square(double i)
```

```
{
```

```
    return i*i;
```

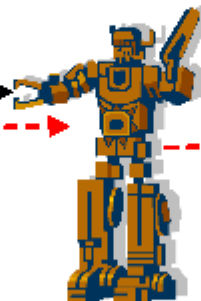
```
}
```



예제

```
int square(int);  
double square(double);
```

```
int main()  
{  
    square(10);  
    square(2.0);  
  
    return 0;  
}
```



```
// 정수값을 제공하는 함수  
int square(int i)  
{  
    return i*i;  
}  
  
// 실수값을 제공하는 함수  
double square(double i)  
{  
    return i*i;  
}
```

그림 4.10 중복 함수의 개념



중복 함수의 장점

- 중복 함수를 사용하지 않은 경우:

```
square_int(int int);
```

```
square_double(double int);
```

```
square_short(short int);
```

- 중복 함수를 사용하는 경우

```
square(int int);
```

```
square(double int);
```

```
square(short int);
```

함수 이름의 재사용이 가능



주의할 점

• `int sub(int);`

• `int sub(int, int);` // 중복 가능!

• `int sub(int, double);` // 중복 가능!

• `double sub(double);` // 중복 가능!

• `double sub(int);` // 오류!! - 반환형이 다르더라도 중복 안됨!

• `float sub(int, int);` // 오류!! - 반환형이 다르더라도 중복 안됨!



인라인 함수

- 인라인 함수(inline function):

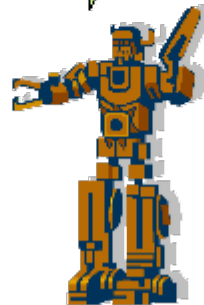
함수 호출을 하지 않고 코드를 복사하여서 넣는 것

inline 인 경우에는
는 함수 몸체를
호출한 곳에 삽입
합니다.

```
int main()
{
    int result = square(10);
    return 0;
}

// 정수값을 제공하는 함수
inline int square(int i)
{
    return i*i;
}
```

```
int main()
{
    int result = 10*10;
    return 0;
}
```





예제



```
#include <iostream>
using namespace std;
// 정수값을 제공하는 함수
inline double square(double i)
{
    return i*i;
}
int main()
{
    double result;
    cout << "2.0의 제곱은 ";
    result = square(2.0);
    cout << result << endl;
    cout << "3.0의 제곱은 ";
    result = square(3.0);
    cout << result << endl;
    return 0;
}
```



2.0의 제곱은 4
3.0의 제곱은 9
계속하려면 아무 키나 누르십시오 . . .



동적 할당 메모리의 개념

- 프로그램이 메모리를 할당받는 방법
 - 정적(static)
 - 동적(dynamic)
- 정적 메모리 할당
 - 프로그램이 시작되기 전에 미리 정해진 크기의 메모리를 할당받는 것
 - 메모리의 크기는 프로그램이 시작하기 전에 결정

```
int i, j;  
int buffer[80];  
char name[] = "data structure";
```

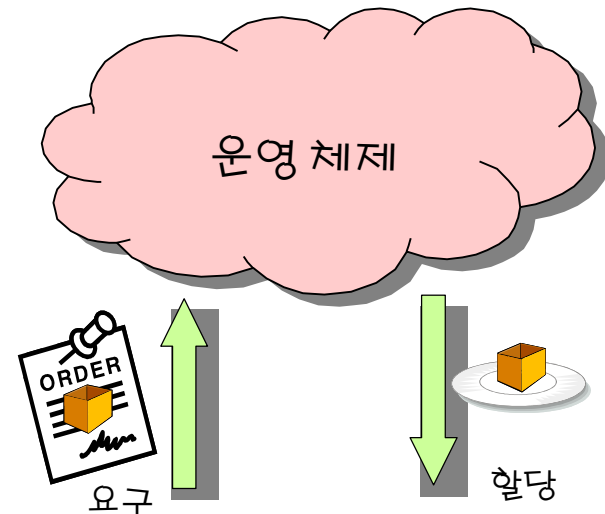
- 처음에 결정된 크기보다 더 큰 입력이 들어온다면 처리하지 못함
- 더 작은 입력이 들어온다면 남은 메모리 공간은 낭비



동적 메모리

- 동적 메모리

- 실행 도중에 동적으로 메모리를 할당받는 것
- 사용이 끝나면 시스템에 메모리를 반납
- 필요한 만큼만 할당을 받고 메모리를 매우 효율적으로 사용
- new와 delete 키워드 사용



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

int main()
{
    int *p;
    p = new int;
    ...
}
```

프로그램



동적 메모리 할당의 과정

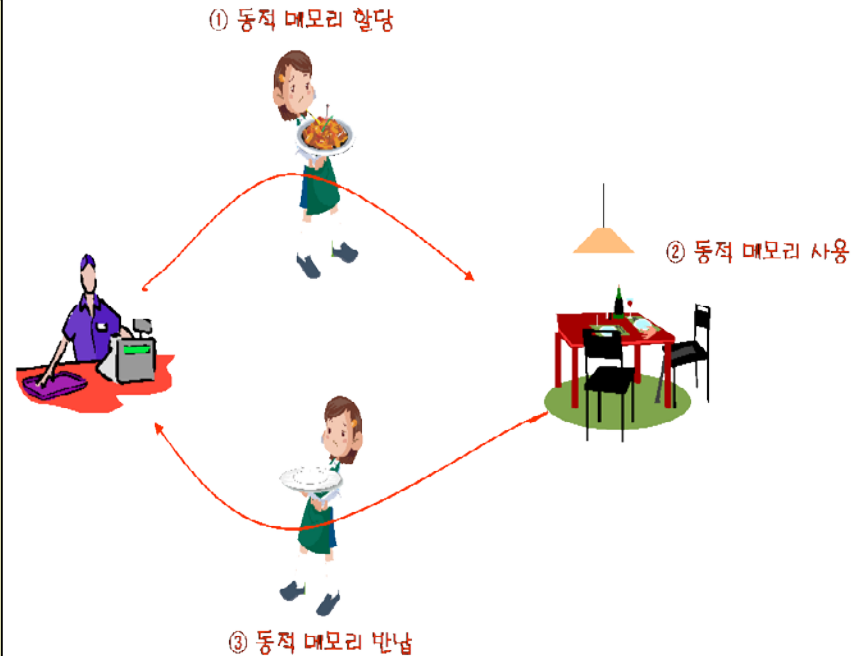
```
#include <iostream>
using namespace std;

int main()
{
    int *pi;        // 동적 메모리를 가리키는 포인터

    pi = new int; // ① 동적 메모리 할당

    *pi = 100;      // ② 동적 메모리 사용
    delete pi;     // ③ 동적 메모리 반납

    return 0;
}
```





파생 자료형인 경우

- 배열

```
double *pd = new double[10];
```

```
...
```

```
delete[] pd;
```




메모리 누수의 예제



```
void sub()
```

```
{
```

```
    int *pi = new int;           // ①
```

```
    *pi = 67;
```

```
    pi = new int;               // ②
```

```
    *pi = 99;
```

```
}
```

잘못된 버전

```
void sub()
```

```
{
```

```
    int *pi = new int;           // ①
```

```
    *pi = 67;
```

```
    delete pi;
```

```
    pi = new int;               // ②
```

```
    *pi = 99;
```

```
    delete pi;
```

```
}
```

올바른 버전



const 포인터

- `const int *p1;`
- `p1`은 `const int`에 대한 포인터이다. 즉 `p1`이 가리키는 내용이 상수가 된다.
- `*p1 = 100;` (X)

- `int * const p2;`
- 이번에는 정수를 가리키는 `p2`가 상수라는 의미이다. 즉 `p2`의 내용이 변경될 수 없다.
- `p2 = p1;` (X)



중간 점검 문제

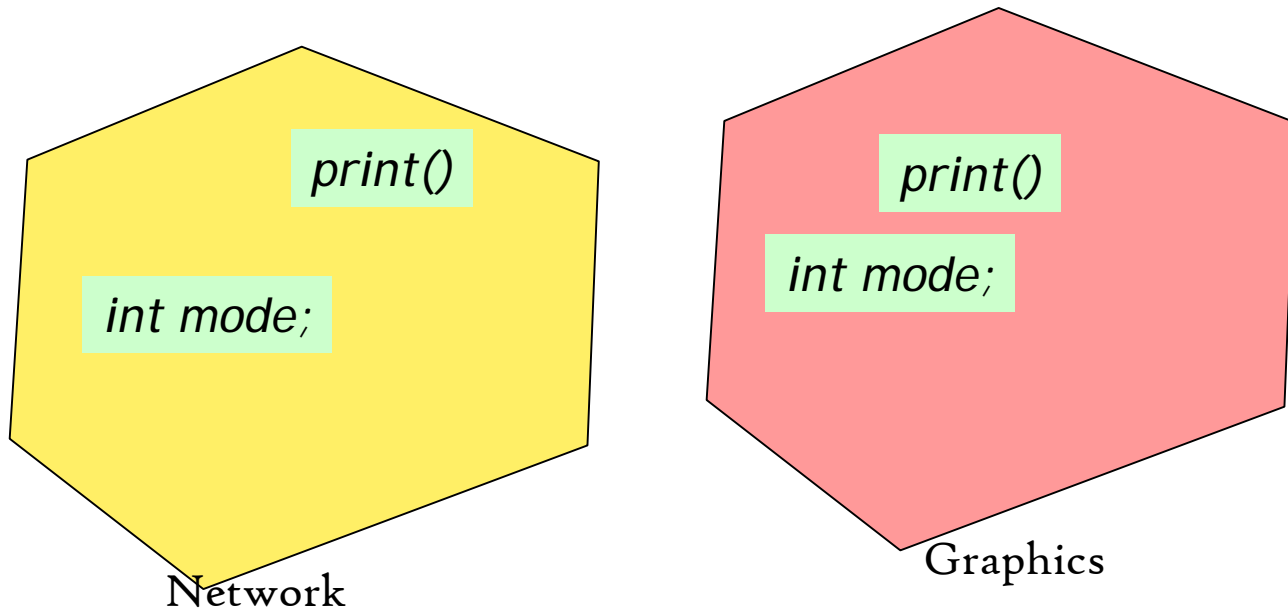
1. 프로그램의 실행 도중에 메모리를 할당받아서 사용하는 것을 _____이라고 한다.
2. 동적으로 메모리를 할당받을 때 사용하는 키워드는 _____이다.
3. 동적으로 할당된 메모리를 해제하는 키워드는 _____이다.





이름 공간

- 이름 공간(name space)는 식별자들이 존재하는 공간
- 이름 공간은 식별자들이 충돌하는 문제를 해결하기 위하여 제안
- 식별자 공간을 논리적으로 분할하고 식별자의 범위를 제한하는 것이 가능





이름 공간 정의

형식

```
namespace 이름 {  
    변수 정의;  
    함수 정의;  
    클래스 정의;  
    ...  
}
```

예

```
namespace Graphics {  
    int mode;  
    int x, y;  
    void draw();  
    void message();  
    ...  
}
```



같은 이름 사용 가능

```
namespace Graphics {  
    int mode;  
    int x, y;  
    void draw();  
    void message();  
    ...  
}
```

```
namespace Network {  
    int mode;  
    int speed;  
    void send(char *);  
    void message();  
    ...  
}
```



이름 공간 지정

- Graphics 이름 공간 안의 mode 변수를 사용하려면

```
Graphics::mode = 1;  
Network::mode = 1;
```

이름 공간 지정



예제



```
#include <iostream>
using namespace std;
int mode; // 전역 변수 mode

namespace Graphics {
    int mode;
    int x, y;
    void draw() { cout << "Graphics 이름 공간 안의 draw()" << endl; };
    void message() { cout << "Graphics 이름 공간 안의 message()" << endl; };
}

namespace Network {
    int mode, speed;
    void send(char *) { cout << "Graphics 이름 공간 안의 send()" << endl; };
    void message() { cout << "Network 이름 공간 안의 message()" << endl; };
}
```




예제



```
int main()
{
    //x = y = 100;           // 컴파일 오류!
    //speed = 22900;         // 컴파일 오류!
    //draw();                // 컴파일 오류!

    mode = 1;               // 전역 변수
    Graphics::mode = 1;
    Network::mode = 2;

    Graphics::message();
    Network::message();
    return 0;
}
```



Graphics 이름 공간 안의 message()
Network 이름 공간 안의 message()
계속하려면 아무 키나 누르십시오 . . .



using 문

```
using 이름공간::식별자;
```

- 예를 들어서 다음과 같이 선언하면 Network안의 mode는 이름 공간을 붙이지 않아도 접근이 가능하다.
- `using Network::speed;`
- `speed = 100; // Network 이름 공간 안의 speed를 의미`

```
using namespace 이름공간;
```

- `using namespace Network;`
- `speed = 19200; // Network 이름 공간 안의 speed를 의미`
- `send("This is a test"); // Network 이름 공간 안의 send()를 의미`



전역 범위 접근

- :: 연산자를 사용한다.
- (예) `::mode=0;`



예제



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

```
int mode; // 전역 변수 mode
void message()
```

```
{
    cout << "전역 공간 안의 message()" << endl;
}
```

```
namespace Graphics {
```

```
    int mode;
```

```
    int x, y;
```

```
    void draw() { cout << "Graphics 이름 공간 안의 draw()" << endl; };
```

```
    void message() { cout << "Graphics 이름 공간 안의 message()" << endl; };
```

```
}
```



예제



```
int main()
{
    using namespace Graphics;

    //mode = 1;           // 컴파일 오류!!
    Graphics::mode = 1;   // Graphics 공간 안의 mode
    ::mode = 1;          // 전역 변수 mode

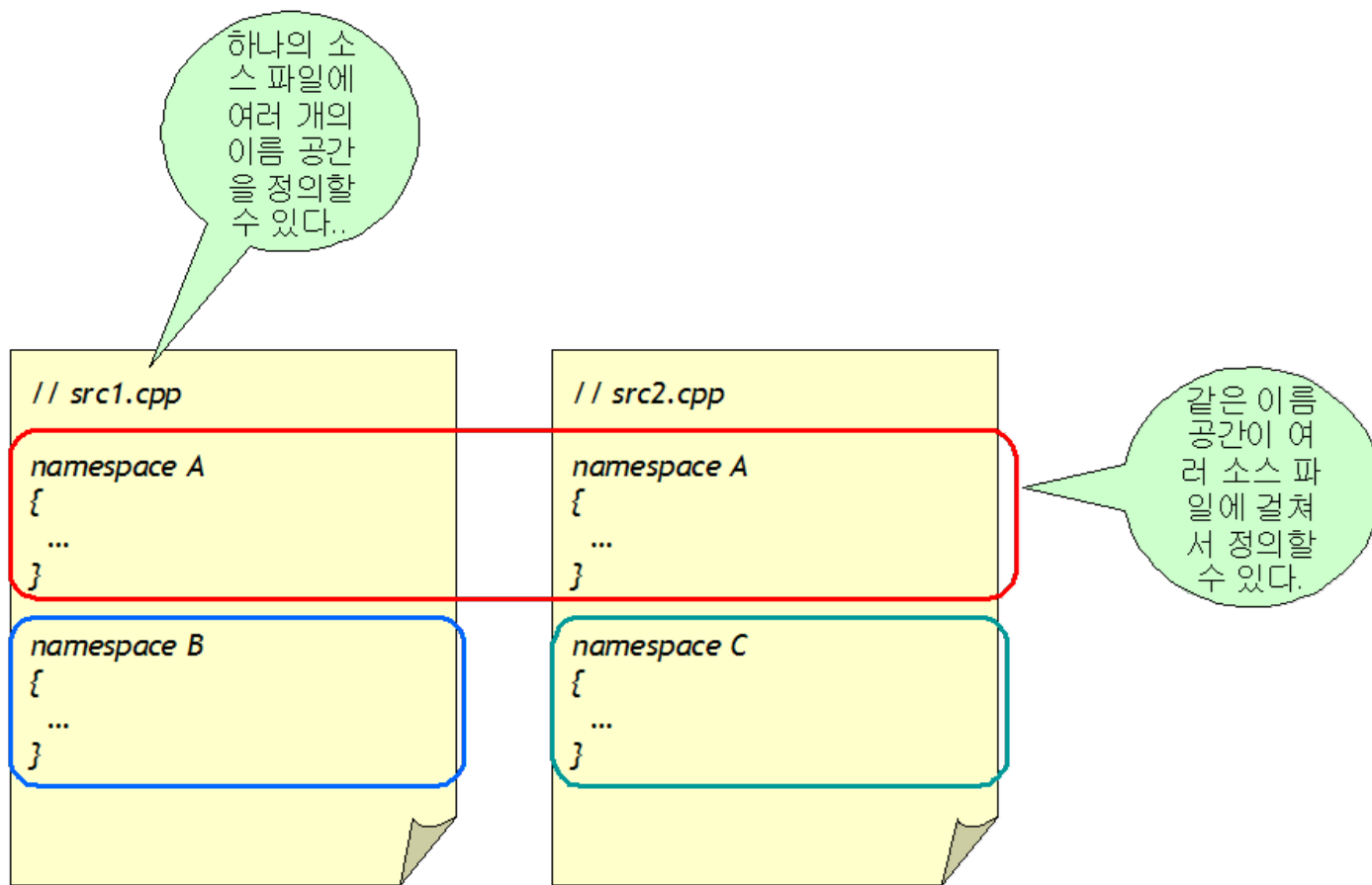
    //message();          // 컴파일 오류
    Graphics::message();  // Graphics 공간 안의 message()
    ::message();          // 전역 함수
    return 0;
}
```



Graphics 이름 공간 안의 **message()**
전역 공간 안의 **message()**
계속하려면 아무 키나 누르십시오 ...



이름 공간과 소스 파일





이름 공간과 헤더 파일

```
// graphics.h
namespace Graphics
{
    extern int x;
    extern int y;
    void draw();
    ...
}
```

이름 공간에 속하는 변수와 함수, 클래스를 정의한다.

```
// graphics.cpp
#include "graphics.h"
```

```
int Graphics::x = 0;
void Graphics::draw()
{
    ...
}
...
```

```
// test.cpp
#include "graphics.h"
```

```
int main()
{
    Graphics::draw();
}
...
```



중간 점검 문제

1. 지역 변수와 같은 이름의 전역 변수를 접근하려면 어떻게 하는가?
2. 이름 공간을 사용하는 이유는 무엇인가?





Q & A

