

객체 Objects

2023

국민대학교 소프트웨어학부

객체들의 배열

- Car carArray[3];

```
1  #include <iostream>
2  using namespace std;
3
4  class Car{
5      string color;
6  public:
7      int speed;
8      Car(int s=0, string c="white"):speed(s), color(c){}
9      void display(){
10         cout << this << " ] " << speed << ", " << color << endl;
11     }
12 };
```

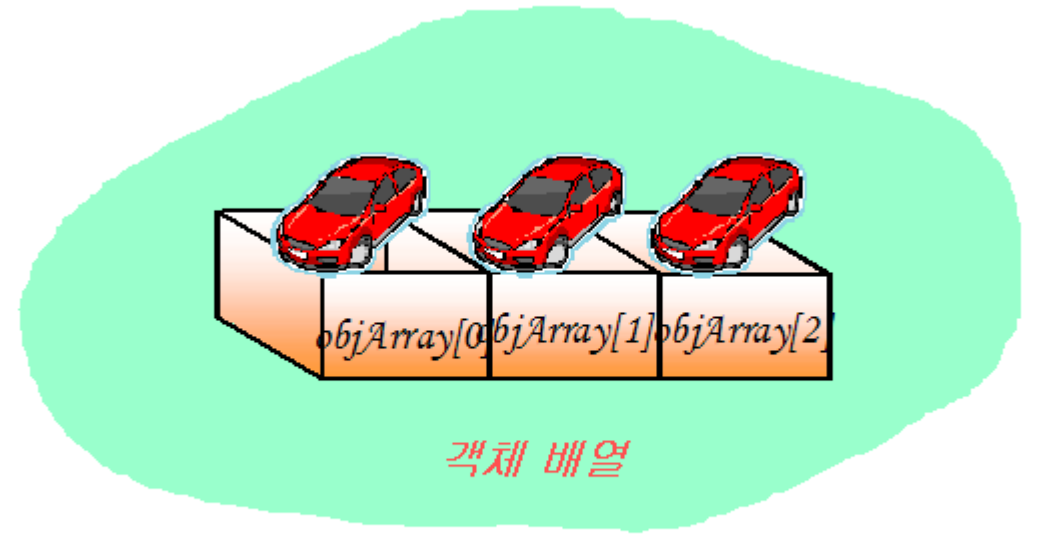


그림 10.8 객체 배열

carArray[0].speed = 0; // 멤버 변수 접근
carArray[1].display(); // 멤버 함수 호출

객체 배열의 초기화

```
Car carArray[3] = {  
    Car(10, "white"),  
    Car(20, "red"),  
    Car(0, "blue"),  
};
```



객체 별로 생성자를 호출
할 수 있다.

```
1  #include <iostream>
2  using namespace std;
3
4  class Car{
5      string color;
6  public:
7      int speed;
8      Car(int s=0, string c="white"):speed(s), color(c){}
9      void display(){
10         cout << this << " ] " << speed << ", " << color << endl;
11     }
12 };
```

```
14  int main(){
15      Car carArray[2] = {
16          Car(100,"red"), Car(50,"blue")};
17      for (int i=0; i<2; i++)
18          carArray[i].display();
19      return 0;
20 }
```

```
0x7ffc7c44d000 ] 100, red
0x7ffc7c44d028 ] 50, blue
```

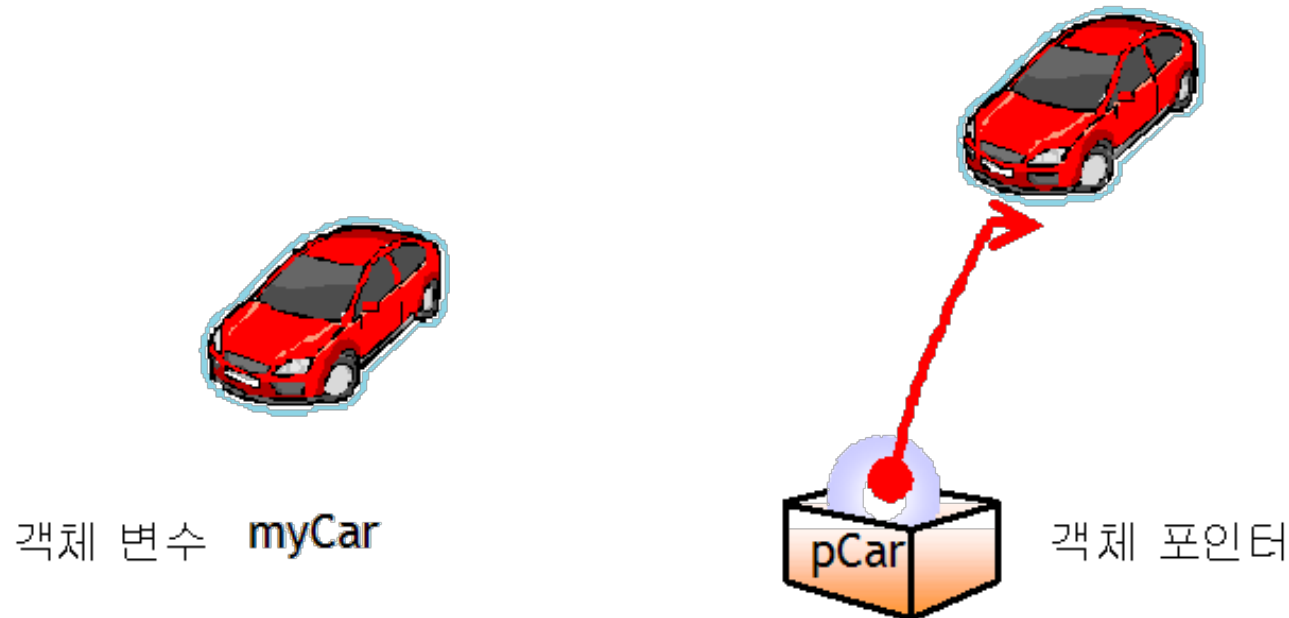
객체의 동적 생성

- 객체도 동적으로 생성할 수 있다.

Car myCar; 생성자의 인자 전달 // 정적 메모리 할당으로 객체 생성

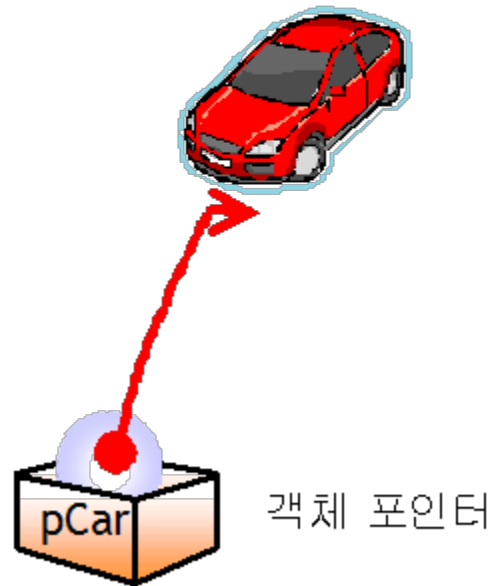
Car *pCar = new Car(); // 동적 메모리 할당으로 객체 생성

pCar = new Car[3]; // 동적으로 객체 배열을 할당받을 때는 생성자의 인자를 줄 수 없다. → default constructor 가 필요하다.



객체 포인터를 통한 멤버 접근

- `pCar->speed = 100;` `// x->y` 는 `(*x).y` 의 뜻
- `pCar->display();`



this 포인터

- this는 현재 코드를 실행하는 객체를 가리키는 포인터

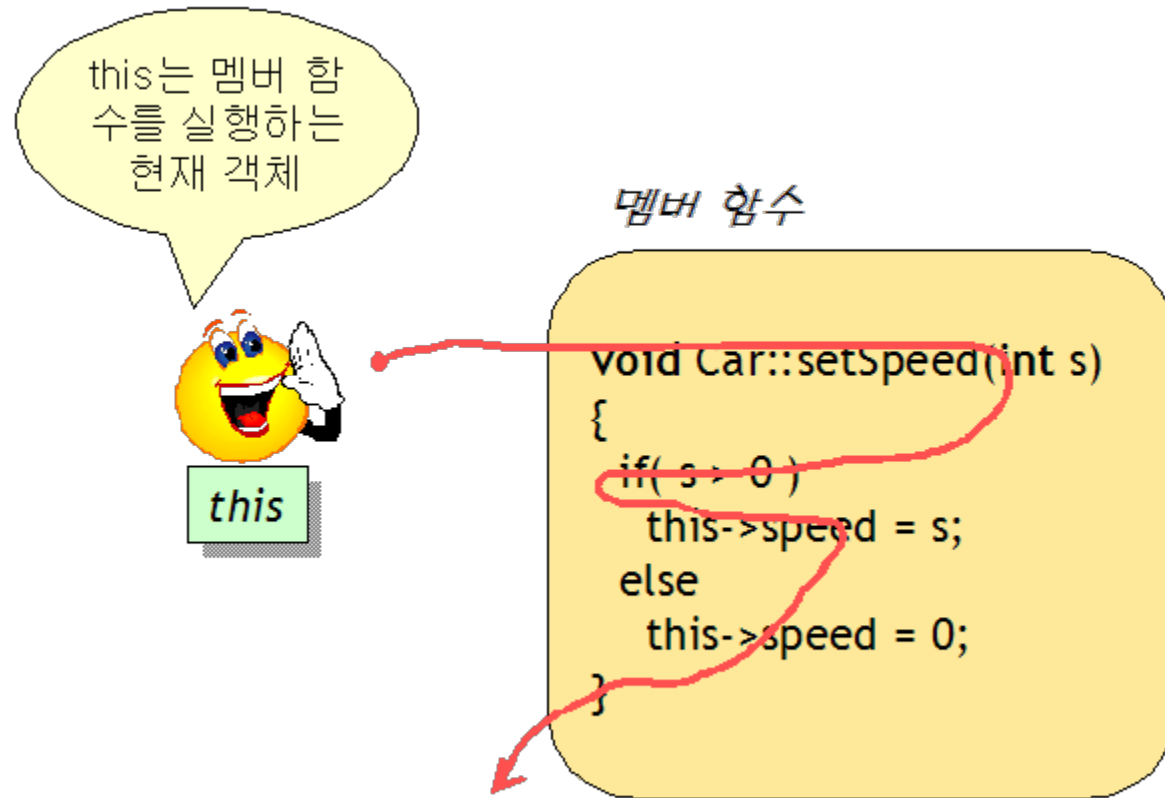


그림 10.2 this 포인터

```

4  class A{
5      int p;
6  public:
7      A(int v = 0){
8          cout << this << " : A(int) called\n";
9          p = v;
10     }
11     ~A(){
12         cout << this << " : ~A() called\n";
13     }
14     int getP(){
15         return p;
16     }
17     void setP(int v){
18         p = (v<0)? 0 : v;
19     }
20 };
21 A a2(2);
22 int main(){
23     A a1;
24     cout << a1.getP() << endl;
25     cout << a2.getP() << endl;
26     return 0;
27 }

```

a1 의 멤버 함수에서 &a1 을 의미하는 pointer
a2 의 멤버 함수에서 &a2 을 의미하는 pointer

```

0x564cc54df134 : A(int) called
0x7fff6787c014 : A(int) called
0
2
0x7fff6787c014 : ~A() called
0x564cc54df134 : ~A() called

```


this 가 필요한 예



```
void Car::setSpeed(int speed)
```

```
{  
    if( speed > 0 )  
        this->speed = speed; // speed는 매개 변수, this->speed는 멤버 변수  
    else  
        this->speed = 0;  
}
```



```
// 생성자
```

```
Car::Car(int s) {  
    this->setSpeed(s); // this는 없어도 된다. 멤버 함수임을 강조  
    this->color = "white";  
}
```

함수에서 객체의 전달

- 1) 객체가 함수의 매개 변수로 전달되는 경우
- 2) 객체의 포인터가 함수의 매개 변수로 전달되는 경우
- 3) 객체의 레퍼런스가 함수의 매개 변수로 전달되는 경우
 - call-by-reference 의 용도 : 함수에서 실인자의 값이 변경됨
 - 인자를 복사하지 않기 위한 용도 : `const A&` 로 선언
- 4) 함수가 객체를 반환하는 경우

1) 객체가 함수의 매개 변수로 전달

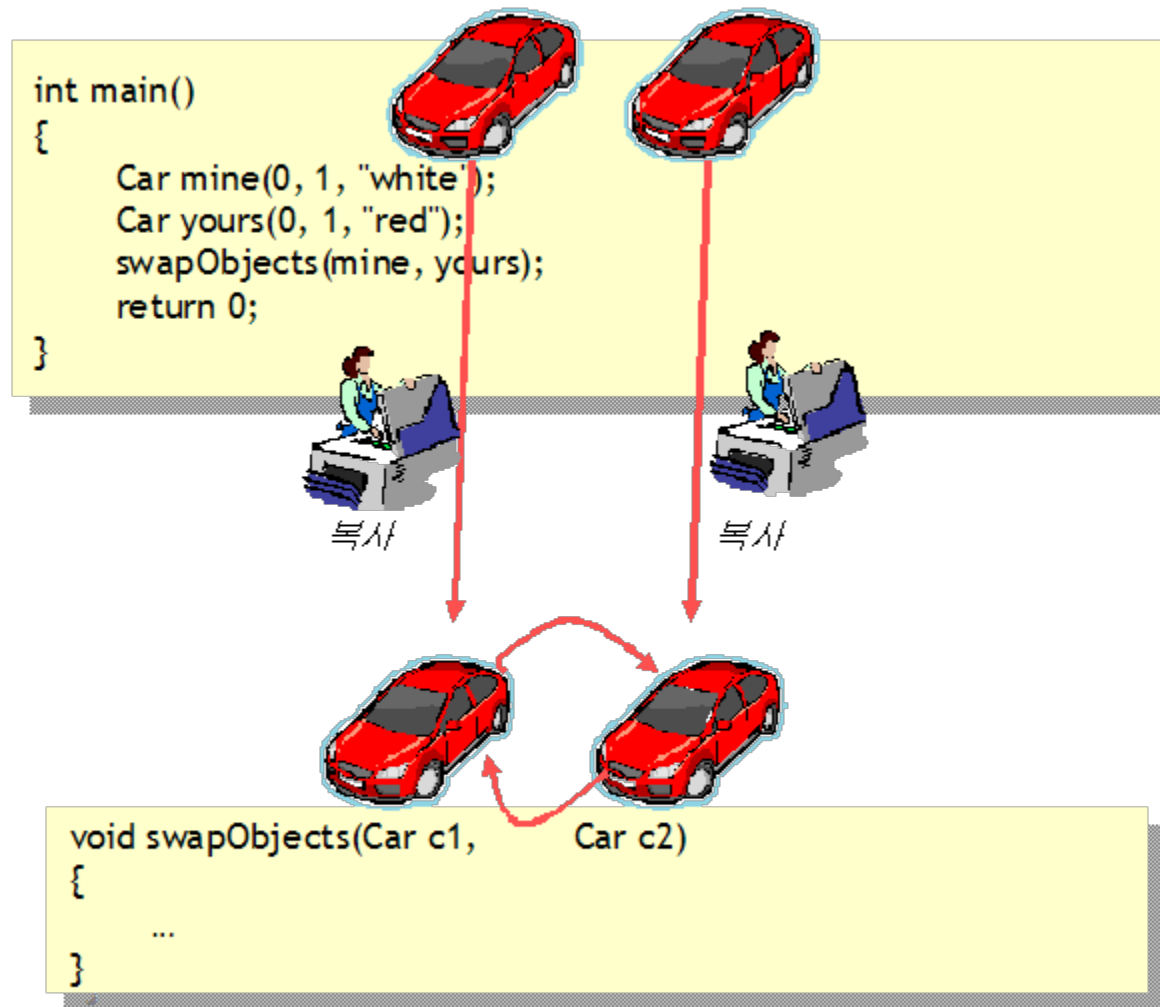


그림 10.4 객체는 값으로 전달된다.

```

4 class Car{
5     int speed;
6     string color = "red";
7 public:
8     Car(int s, string c):speed(s), color(c){}
9     void display(){
10         cout << this << " ] " << speed << ", " << color << endl;
11     }
12 };
13
14 void swapObjects(Car c1, Car c2){
15     Car tmp;
16     tmp = c1;
17     c1 = c2;
18     c2 = tmp;
19 }
20
21 int main(){
22     Car redCar(100,"red"), blueCar(50,"blue");
23     redCar.display(); blueCar.display();
24     swapObjects(redCar, blueCar);
25     redCar.display(); blueCar.display();
26     return 0;
27 }

```

```

g++ -g -o car2 car2.cpp
car2.cpp: In function 'void swapObjects(Car, Car)':
car2.cpp:15:7: error: no matching function for call to 'Car::Car()'
    Car tmp;
    ^~~
car2.cpp:8:3: note: candidate: Car::Car(int, std::__cxx11::string)
    Car(int s, string c):speed(s), color(c){}
    ^~~
car2.cpp:8:3: note: candidate expects 2 arguments, 0 provided
car2.cpp:4:7: note: candidate: Car::Car(const Car&)
    class Car{
    ^~~
car2.cpp:4:7: note: candidate expects 1 argument, 0 provided
car2.cpp:4:7: note: candidate: Car::Car(Car&&)
car2.cpp:4:7: note: candidate expects 1 argument, 0 provided
Makefile:19: recipe for target 'car2' failed
make: *** [car2] Error 1

```

```

4 class Car{
5     int speed;
6     string color = "red";
7 public:
8     Car(int s=0, string c="white"):speed(s), color(c){}
9     void display(){
10         cout << this << " ] " << speed << ", " << color << endl;
11     }
12 };
13
14 void swapObjects(Car c1, Car c2){
15     Car tmp;
16     tmp = c1;
17     c1 = c2;
18     c2 = tmp;
19 }
20
21 int main(){
22     Car redCar(100,"red"), blueCar(50,"blue");
23     redCar.display(); blueCar.display();
24     swapObjects(redCar, blueCar);
25     redCar.display(); blueCar.display();
26     return 0;
27 }

```

default constructor 를 만들어 준다.

Function	Variable	Address	Value
main()	blueCar	0x7ffc349e3fc0	50, blue
	redCar	0x7ffc349e3f90	100, red
swapObjects()	c1	0x7ffc349e3f60	100, red
	c2	0x7ffc349e3f30	50, blue
	tmp	0x7ffc349e3f00	0, white

```

redCar 0x7ffc349e3f90 ] 100, red
blueCar 0x7ffc349e3fc0 ] 50, blue
redCar 0x7ffc349e3f90 ] 100, red
blueCar 0x7ffc349e3fc0 ] 50, blue

```

2) 객체의 포인터가 함수의 매개변수로 전달

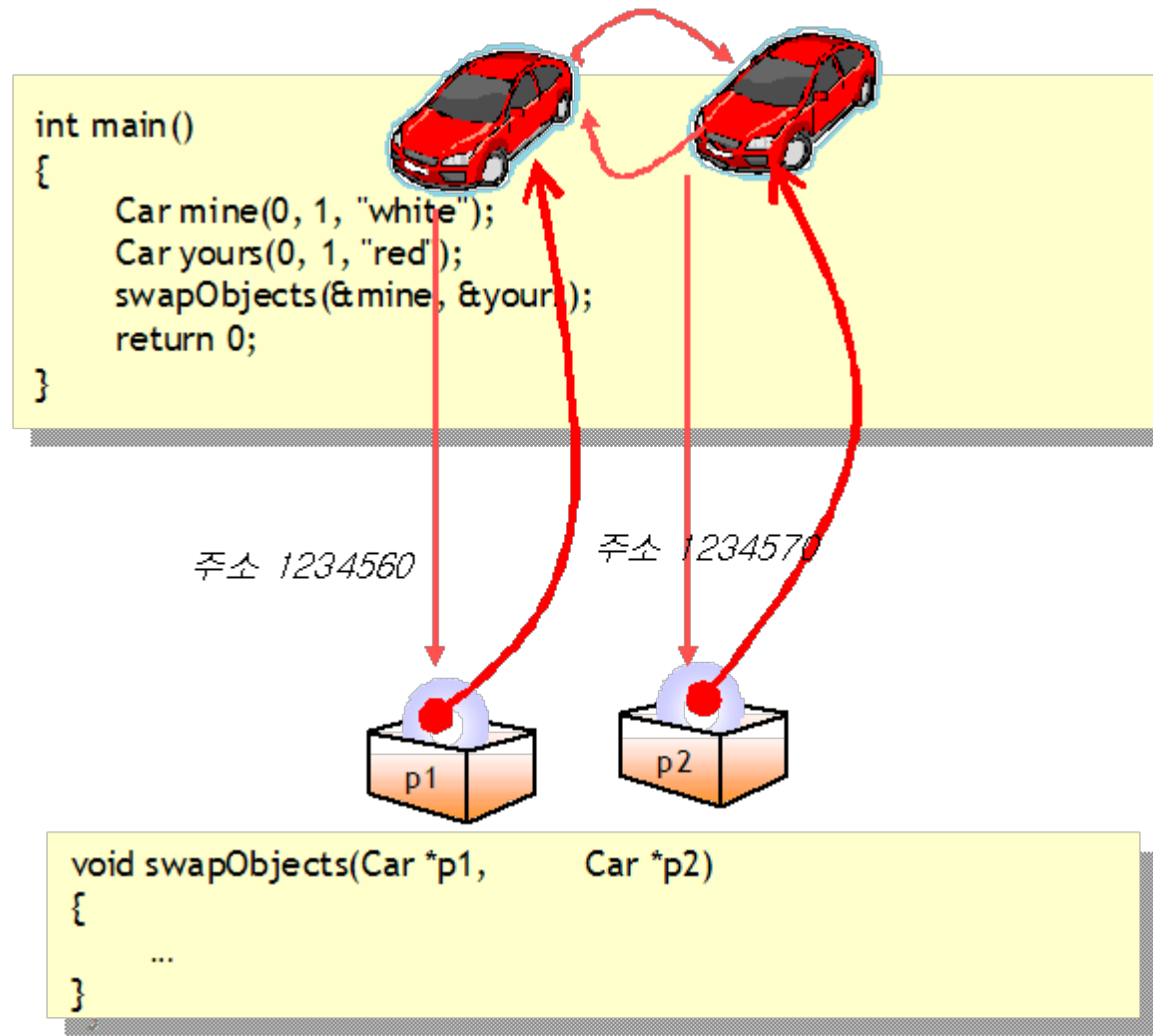


그림 10.5 객체의 포인터를 전달하는 경우

```

4  class Car{
5      int speed;
6      string color;
7  public:
8      Car(int s=0, string c="white"):speed(s), color(c){}
9      void display(){
10         cout << this << " ] " << speed << ", " << color << endl;
11     }
12 };
13
14 void swapObjectsPtr(Car *c1, Car *c2){
15     Car tmp;
16     tmp = *c1;
17     *c1 = *c2;
18     *c2 = tmp;
19 }
20
21 int main(){
22     Car redCar(100,"red"), blueCar(50,"blue");
23     redCar.display(); blueCar.display();
24     swapObjectsPtr(&redCar, &blueCar);
25     redCar.display(); blueCar.display();
26     return 0;
27 }

```

redC
 blueC
 redC
 blueC

[illegible]

3) 객체의 레퍼런스가 함수의 매개변수로 전달

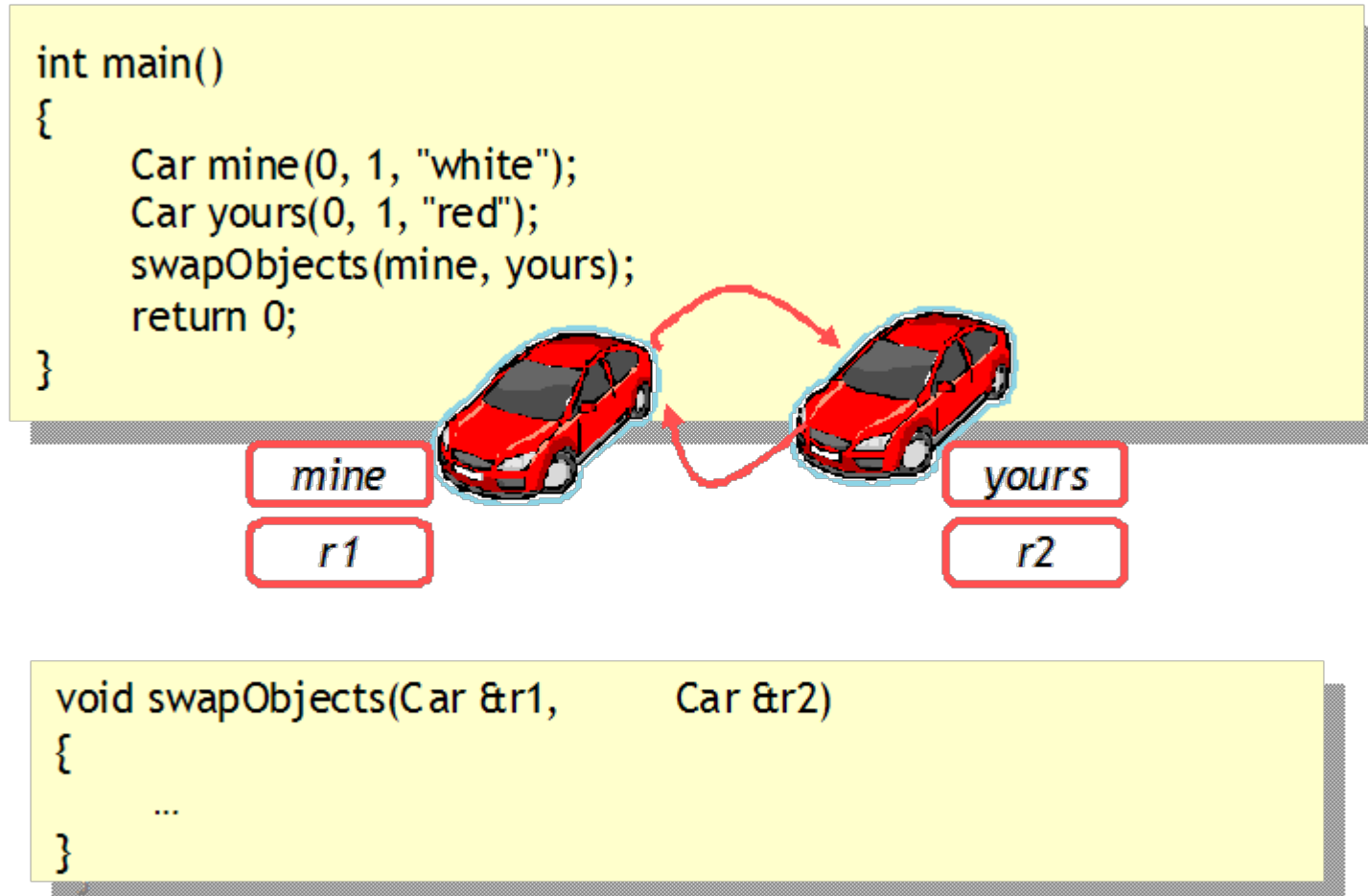


그림 10.6 객체의 레퍼런스를 전달하는 경우


```

4 class Car{
5     int speed;
6     string color;
7 public:
8     Car(int s=0, string c="white"):speed(s), color(c){}
9     void display(){
10         cout << this << " ] " << speed << ", " << color << endl;
11     }
12 };
13
14 void swapObjectsRef(Car& c1, Car& c2){
15     Car tmp;
16     tmp = c1;
17     c1 = c2;
18     c2 = tmp;
19 }
20
21 int main(){
22     Car redCar(100,"red"), blueCar(50,"blue");
23     redCar.display(); blueCar.display();
24     swapObjectsRef(redCar, blueCar);
25     redCar.display(); blueCar.display();
26     return 0;
27 }

```

main()	blueCar	0x7ffed9e22940	50, blue	(c2)
	redCar	0x7ffed9e22910	100, red	(c1)
swapObjectsRef()	tmp		0, white	

```

redCar 0x7ffed9e22910 ] 100, red
blueCar 0x7ffed9e22940 ] 50, blue
redCar 0x7ffed9e22910 ] 50, blue
blueCar 0x7ffed9e22940 ] 100, red

```

• 함수가 객체를 반환

```
4  class Car{
5      int speed;
6      string color;
7  public:
8      Car(int s=0, string c="white"):speed(s), color(c){}
9      void display(){
10         cout << this << " ] " << speed << ", " << color << endl;
11     }
12 };
13
14 Car buyCar(string color){
15     Car tmp(30, color);
16     tmp.display();
17     return tmp;
18 }
19
20 int main(){
21     Car myCar;
22     myCar.display();
23     myCar = buyCar("green");
24     myCar.display();
25     return 0;
26 }
```

built-in type 과 비교

```
4  int buyInt(){
5      int i;
6      i = 30;
7      return i;
8  }
9
10 int main(){
11     int r;
12
13     r = buyInt();
14     return 0;
15 }
```

myCar	0x7ffdebd50620] 0, white
tmp	0x7ffdebd50650] 30, green
myCar	0x7ffdebd50620] 30, green

객체와 연산자

- 객체에 할당 연산자(=)를 사용할 수 있는가?

```
class Car
{
    ... //생략
};
```

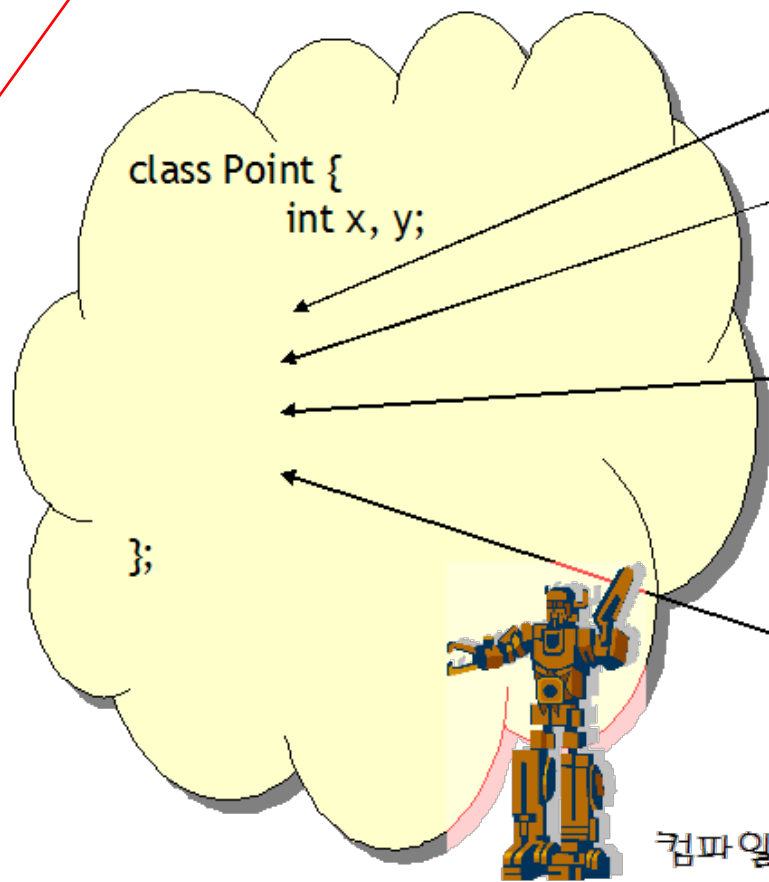
```
int main()
{
    Car c1(0, 1, "white");
    Car c2(0, 1, "red");
    c1 = c2; // 어떻게 되는가?
    return 0;
}
```

c2 객체가 가지고 있는
변수의 값이 c1으로 복사
된다..

디폴트 멤버 함수

- 디폴트 생성자
- 디폴트 소멸자
- 디폴트 복사 생성자
- 디폴트 할당 연산자

자동으로 추가된다.



디폴트 생성자

```
Point() { }
```

디폴트 소멸자

```
~Point() { }
```

디폴트 복사 생성자

```
Point(Point& p)
{
    x = p.x; y = p.y;
}
```

디폴트 할당 연산자

```
Point& operator=(Point& p)
{
    x = p.x; y = p.y;
    return *this;
}
```

컴파일러

그림 10.7 디폴트 멤버 함수의 추가

```

4  class Car{
5      string color;
6  public:
7      int speed;
8      Car(int s=0, string c="white"):speed(s), color(c){}
9      void display(){
10         cout << this << " ] " << speed << ", " << color << endl;
11     }
12 };
13
14 int main(){
15     Car carArray[2] = {
16         Car(100,"red"), Car(50,"blue")};
17     for (int i=0; i<2; i++)
18         carArray[i].display();
19
20     cout<< "sizeof(Car) = " << sizeof(Car) << endl;
21     Car *ptr = new Car[3]; // Car *ptr; ptr=new Car[3];
22     // cout << *ptr == *(ptr+1) << endl; ← compile error
23     ptr->display();
24     (ptr+1)->display();
25     *ptr = carArray[0];
26     ptr->display();
27 }

```

carArray[0]	0x7ffed1e71600]	100, red
carArray[1]	0x7ffed1e71628]	50, blue
sizeof(Car) = 40		
*ptr	0x558fb3662288]	0, white
*(ptr+1)	0x558fb36622b0]	0, white
*ptr	0x558fb3662288]	100, red

객체와 연산자

- 객체에 비교 연산자(==)를 사용할 수 있는가?

```
class Car
{
    ... //생략
};

int main()
{
    Car c1(0, 1, "white");
    Car c2(0, 1, "red");
    if( c1 == c2 ){
        cout << "같습니다" << endl;
    }
    else {
        cout << "같습니다" << endl;
    }
    return 0;
}
```

연산자 중복이 되
어 있지 않으면
compiler error!
-> 뒤에 학습

```
4  class Car{
5      string color;
6  public:
7      int speed;
8      Car(int s=0, string c="white"):speed(s), color(c){}
9      void display(){
10         cout << this << " ] " << speed << ", " << color << endl;
11     }
12 };
13
14 int main(){
15     Car carArray[2] = {
16         Car(100,"red"), Car(50,"blue")};
17     for (int i=0; i<2; i++)
18         carArray[i].display();
19
20     cout<< "sizeof(Car) = " << sizeof(Car) << endl;
21     Car *ptr = new Car[3]; // Car *ptr; ptr=new Car[3];
22     // cout << *ptr == *(ptr+1) << endl;
23     ptr->display();
24     (ptr+1)->display();
25     *ptr = carArray[0];
26     ptr->display();
27 }
```

정적 멤버

- 인스턴스 변수(instance variable): 객체마다 하나씩 있는 변수
- 정적 변수(**static** variable): 모든 객체를 통틀어서 하나만 있는 변수

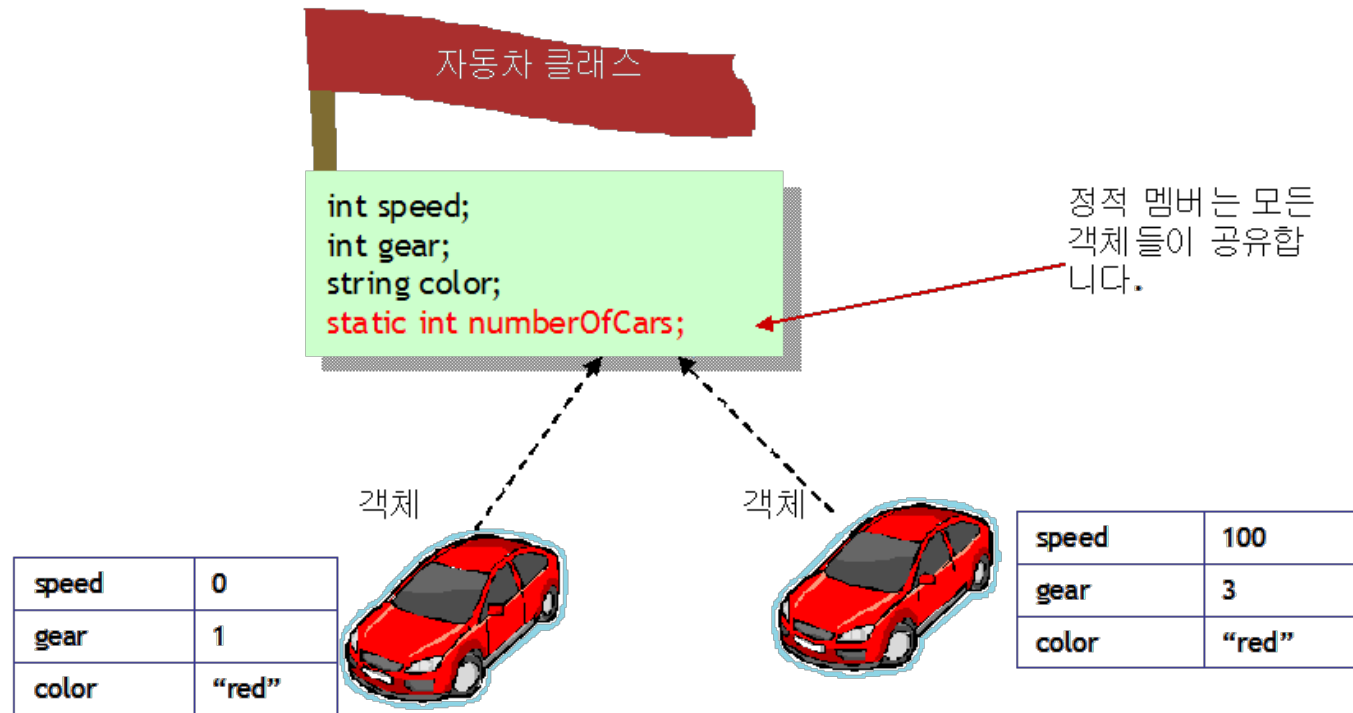


그림 10.7 정적 멤버

정적 멤버 변수

```
4 class Car{
5     string color;
6     int serial;
7 public:
8     int speed;
9     static int numberOfCars;
10    Car(int s=0, string c="white"):speed(s), color(c){
11        serial = ++numberOfCars;
12    }
13    void display(){
14        cout << this << " ] " << serial << ", ";
15        cout << speed << ", " << color << endl;
16    }
17 };
18 int Car::numberOfCars = 0;
```

static member variable 의 초기화는 class 밖에서 해주어야 함
이 변수는 클래스의 객체가 없어도 존재함

```
19 int main(){
20     Car carArray[2] = {
21         Car(100,"red"), Car(50,"blue")};
22     for (int i=0; i<2; i++)
23         carArray[i].display();
24     cout << "sizeof(Car) = " << sizeof(Car) << endl;
25     Car *ptr = new Car[3]; // Car *ptr; ptr=new Car[3];
26     ptr->display();
27     (ptr+1)->display();
28     *ptr = carArray[0];
29     ptr->display();
```

carArray[0]
carArray[1]

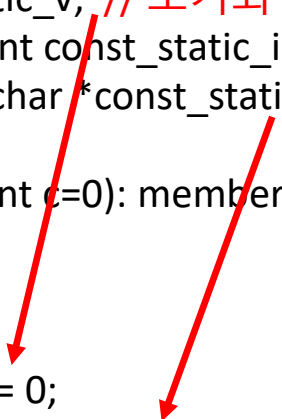
*ptr
*(ptr+1)
*ptr

```
0x7fff22d272a0 ] 1, 100, red
0x7fff22d272c8 ] 2, 50, blue
sizeof(Car) = 40
0x5643aed60288 ] 3, 0, white
0x5643aed602b0 ] 4, 0, white
0x5643aed60288 ] 1, 100, red
```


주의) 정적 멤버 변수의 초기화

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

```
class Car {
    int member_v;
    const int const_int;
    static int static_v; // 초기화 리스트로 초기화할 수 없음
    const static int const_static_int = 1; // integral type (정수 혹은 enumeration) 만 가능
    const static char *const_static_ptr;
public:
    Car(int v=0, int c=0): member_v(v), const_int(c) {
    }
};
```



```
int Car::static_v = 0;
const char *Car::const_static_ptr = &p;
```

정적 멤버 함수

```
4 class Car{
5     string color;
6     int serial;
7 public:
8     int speed;
9     static int numberOfCars;
10    Car(int s=0, string c="white"):speed(s), color(c){
11        serial = ++numberOfCars;
12    }
13    static int getNumberOfCars(){ return numberOfCars;}
14    void display(){
15        cout << this << " ] " << serial << ", ";
16        cout << speed << ", " << color << endl;
17    }
18 };
19 int Car::numberOfCars = 0;
20 int main(){
21     Car carArray[10];
22     for (int i=0; i<10; i++)
23         carArray[i].display();
24     Car *ptr = new Car[30];
25     cout << "Number of Cars is " << Car::getNumberOfCars() << endl;
26     cout << "Number of Cars is " << carArray[2].getNumberOfCars() << endl;
27     cout << "Number of Cars is " << ptr->getNumberOfCars() << endl;
```

객체 없이 호출될 수 있다.

instance member variable 을 사용할 수 없다.

instance member function을 호출할 수 없다.

```
0x7fff0cca55e0 ] 1, 0, white
0x7fff0cca5608 ] 2, 0, white
0x7fff0cca5630 ] 3, 0, white
0x7fff0cca5658 ] 4, 0, white
0x7fff0cca5680 ] 5, 0, white
0x7fff0cca56a8 ] 6, 0, white
0x7fff0cca56d0 ] 7, 0, white
0x7fff0cca56f8 ] 8, 0, white
0x7fff0cca5720 ] 9, 0, white
0x7fff0cca5748 ] 10, 0, white
Number of Cars is 40
Number of Cars is 40
Number of Cars is 40
```

static keyword

- storage class

- static local variable :

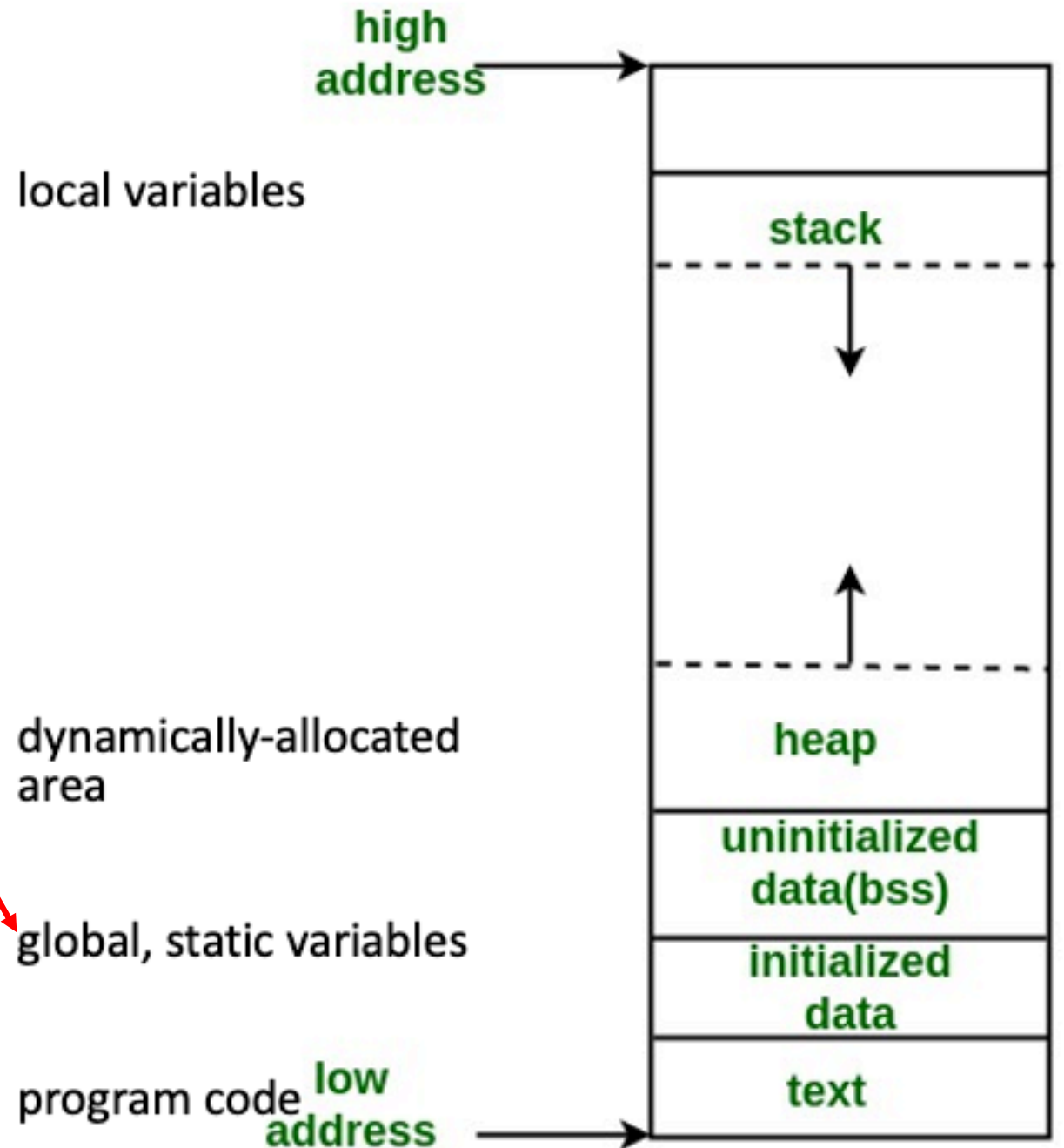
지역 변수를 정적으로 할당

- static global variable / function :

선언된 화일 외부에서 접근 불가

- static member variable :

클래스 공통으로 공유하는 변수

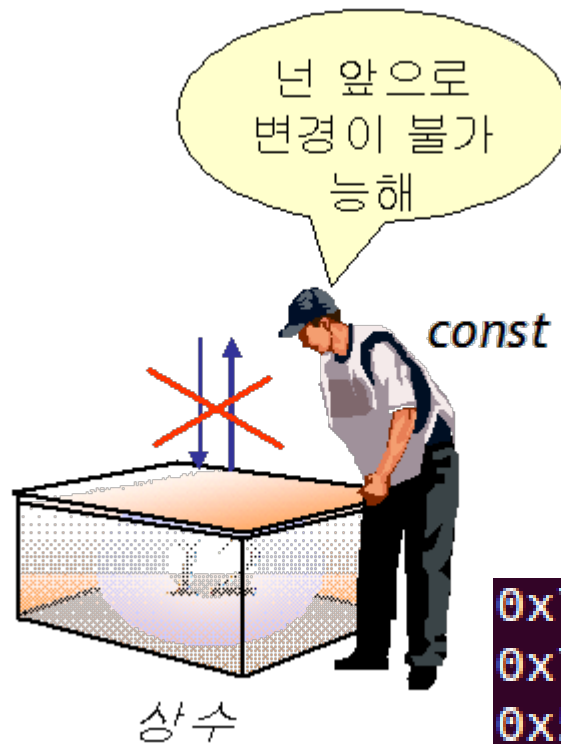


const member variable

- 초기화 방법

- 선언문에서 : class-wide symbolic constant
- 생성자의 초기화 리스트에서 : symbolic constant 의 값을 object 마다 다르게 할 수 있음

```
4  class A{
5      const int c;
6      const int hundred = 100;
7  public:
8      A(int n=0): c(n){}
9      void display(){
10         cout << this << " ] " << c << ", ";
11         cout << hundred << endl;
12     }
13 };
14 int main(){
15     A arr[2] = {A(1), A(2)};
16     for (int i=0; i<2; i++)
17         arr[i].display();
18     A *ptr = new A[3];
19     for (int i=0; i<3; i++)
20         (ptr+i)->display();
21 }
```



0x7ffd2e876e00] 1, 100
0x7ffd2e876e08] 2, 100
0x564c493fa280] 0, 100
0x564c493fa288] 0, 100
0x564c493fa290] 0, 100

const member function

```
4  class A{
5      const int c;
6      const int hundred = 100;
7      int v;
8  public:
9      A(int n=0): c(n){}
10     void display() const {
11         v = 0;
12         cout << this << " ] " << c << ", ";
13         cout << hundred << endl;
14     }
15 };
```

이 함수 안에서는 멤버
변수의 값을 변경할 수
없다.

```
g++ -g -o constfunction constfunction.cpp
constfunction.cpp: In member function 'void A::display() const':
constfunction.cpp:11:10: error: assignment of member 'A::v' in read-only object
    v = 0;
    ^
Makefile:19: recipe for target 'constfunction' failed
make: *** [constfunction] Error 1
```

Why const function?

- const 선언된 object 에 대해서는 const member function 만 호출 가능
- 실제로 member 변수의 값을 바꾸지 않는 함수라도 const 선언이 되어 있지 않으면 호출 불가능

```
g++ -g -o constfunction constfunction.cpp
constfunction.cpp: In function 'int main()':
constfunction.cpp:18:17: error: passing 'const A' as 'this' arg
alifiers [-fpermissive]
    cA.do_nothing();
        ^
constfunction.cpp:14:8: note:   in call to 'void A::do_nothing(
    void do_nothing(){}
        ^~~~~~
Makefile:19: recipe for target 'constfunction' failed
make: *** [constfunction] Error 1
```

```
4  class A{
5      const int c;
6      const int hundred = 100;
7      int v;
8  public:
9      A(int n=0): c(n){}
10     void display() const {
11         cout << this << " ] " << c << ", ";
12         cout << hundred << endl;
13     }
14     void do_nothing(){}
15 };
16 int main(){
17     const A cA;
18     cA.do_nothing();
19     cA.display();
20 }
```

const 수식어도 function signature 에 포함 → 중복 가능

```
4 class A{
5     const int c;
6     const int hundred = 100;
7     int v;
8 public:
9     A(int n=0): c(n){}
10    void display() const {
11        cout << this << " ] " << c << ", ";
12        cout << hundred << " const display()\n";
13    }
14    void display() {
15        cout << this << " ] " << c << ", ";
16        cout << hundred << " display()\n";
17    }
18 };
19 int main(){
20     const A cA(5);
21     A obj(10);
22     cA.display();
23     obj.display();
```

function overloading

```
0x7ffe6e8ba850 ] 5, 100 const display()
0x7ffe6e8ba85c ] 10, 100 display()
```

```

1  #include <iostream>
2  #include <cstring>
3  using namespace std;
4
5  class MyString{
6      char *p;
7  public:
8      MyString(const char *str=NULL);
9      MyString(MyString& s);
10     ~MyString();
11     void print();
12     int size();
13 };
14 MyString::MyString(const char *str){
15     if (!str){
16         p = new char[1];
17         p[0] = '\\0';
18         return;
19     }
20     p = new char[strlen(str)+1];
21     strcpy(p, str);
22 }
23 MyString::MyString(MyString& s){
24     p = new char[s.size()+1]; // deep copy
25     strcpy(p, s.p);
26 }
27 MyString::~MyString(){ delete[] p; }
28 void MyString::print(){ cout << p << endl; }
29 int MyString::size(){ return strlen(p); }

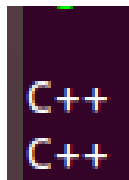
```

in MyString class

```

30 int main(){
31     MyString s1;
32     MyString s2("C++");
33     MyString s3(s2);
34     s1.print();
35     s2.print();
36     s3.print();
37     return 0;
38 }

```



const 포인터

- `const int *p1;`
 - p1은 `const int`에 대한 포인터이다. 즉 p1이 가리키는 내용이 상수가 된다.
 - `*p1 = 100;` (X)
-
- `int * const p2;`
 - 이번에는 정수를 가리키는 p2가 상수라는 의미이다. 즉 p2의 내용이 변경될 수 없다.
 - `p2 = p1;` (X)

```

4 void f(int *i){}
5 void g(int &i){}
6
7 int main(){
8     int v;
9     f(&v);
10    g(v);
11    const int one = 1;
12    f(&one);
13    g(one);
14    return 0;
15 }

```

상수에 대한 레퍼런스/포인터를
const 선언이 되지 않은 레퍼런스/포인터에
대입하는 것은 compile error

→ int *i=&one;
int &i = one;

const 로 선언된 레퍼런스/포인터에는
변수/상수에 대한 레퍼런스/포인터를
대입할 수 있음

const int *i=&one;
const int &i = one;

```

4 void f(const int *i){}
5 void g(const int &i){}
6
7 int main(){
8     int v;
9     f(&v);
10    g(v);
11    const int one = 1;
12    f(&one);
13    g(one);
14    return 0;
15 }

```

```

g++ -g -o c c.cpp
c.cpp: In function 'int main()':
c.cpp:12:5: error: invalid conversion from 'const int*' to 'int*' [-fpermissive]
    f(&one);
    ^~~~
c.cpp:4:6: note: initializing argument 1 of 'void f(int*)'
    void f(int *i){}
    ^
c.cpp:13:8: error: binding reference of type 'int&' to 'const int' discards qual
ifiers
    g(one);
    ^
c.cpp:5:6: note: initializing argument 1 of 'void g(int&)'
    void g(int &i){}
    ^
Makefile:19: recipe for target 'c' failed
make: *** [c] Error 1

```

```

5  class MyString{
6      char *p;
7  public:
8      MyString(const char *str=NULL);
9      MyString(char *str=NULL);
10     MyString(MyString& s);
11     ~MyString(){ delete[] p; }
12     void print() const { cout << p << endl; }
13     int size(){ return strlen(p); }
14 };
15 MyString::MyString(char *str){
16     cout << this << " " << str << "]" MyString(char *)\n";
17 }
18 MyString::MyString(const char *str){
19     if (!str){
20         p = new char[1];
21         p[0] = '\0';
22         return;
23     }
24     p = new char[strlen(str)+1];
25     strcpy(p, str);
26     cout << this << " " << str << "]" MyString(const char *)\n";
27 }
28 MyString::MyString(MyString& s){
29     p = new char[s.size()+1]; // deep copy
30     strcpy(p, s.p);
31     cout << this << " " << s.p << "]" MyString(MyString&)\n";
32 }

```

중복 가능

```

34 int main(){
35     const MyString s1("const");
36     MyString s2("C++");
37     MyString s3(s2);
38     s1.print();
39     s2.print();
40     s3.print();
41     return 0;
42 }

```

```

s1 0x7fff8e5e19a0 const] MyString(const char *)
s2 0x7fff8e5e19a8 C++] MyString(const char *)
s3 0x7fff8e5e19b0 C++] MyString(MyString&)
s1 const
s2 C++
s3 C++

```

```

5  class MyString{
6      char *p;
7  public:
8      MyString(const char *str=NULL);
9      MyString(char *str=NULL);
10     MyString(MyString& s);
11     ~MyString(){ delete[] p; }
12     void print() const { cout << p << endl; }
13     int size(){ return strlen(p); }
14 };
15 MyString::MyString(char *str){
16     cout << this << " " << str << "]" MyString(char *)\n";
17 }
18 MyString::MyString(const char *str){
19     if (!str){
20         p = new char[1];
21         p[0] = '\0';
22         return;
23     }
24     p = new char[strlen(str)+1];
25     strcpy(p, str);
26     cout << this << " " << str << "]" MyString(const char *)\n";
27 }
28 MyString::MyString(MyString& s){
29     p = new char[s.size()+1]; // deep copy
30     strcpy(p, s.p);
31     cout << this << " " << s.p << "]" MyString(MyString&)\n";
32 }

```

```

34 int main(){
35     const MyString s1("const");
36     MyString s2("C++");
37     MyString s3(s1);
38     s1.print();
39     s2.print();
40     s3.print();
41     return 0;
42 }

```

```

g++ -g -o MyString1 MyString1.cpp
MyString1.cpp: In function 'int main()':
MyString1.cpp:37:17: error: binding reference of type 'MyString&' to 'const MyString' discards qualifiers
    MyString s3(s1);
                  ^
MyString1.cpp:28:1: note: initializing argument 1 of 'MyString::MyString(MyString&)'
    MyString::MyString(MyString& s){
    ~~~~~~
Makefile:19: recipe for target 'MyString1' failed
make: *** [MyString1] Error 1

```

```

5 class MyString{
6     char *p;
7 public:
8     MyString(const char *str=NULL);
9     MyString(char *str=NULL);
10    MyString(const MyString& s);
11    ~MyString(){ delete[] p; }
12    void print() const { cout << p << endl; }
13    int size(){ return strlen(p); }
14 };
15 MyString::MyString(char *str){
16     cout << this << " " << str << "]" MyString(char *)\n";
17 }
18 MyString::MyString(const char *str){
19     if (!str){
20         p = new char[1];
21         p[0] = '\0';
22         return;
23     }
24     p = new char[strlen(str)+1];
25     strcpy(p, str);
26     cout << this << " " << str << "]" MyString(const char *)\n";
27 }
28 MyString::MyString(const MyString& s){
29     p = new char[s.size()+1]; // deep copy
30     strcpy(p, s.p);
31     cout << this << " " << s.p << "]" MyString(const MyString&)\n";
32 }

```

```

34 int main(){
35     const MyString s1("const");
36     MyString s2("C++");
37     MyString s3(s1);
38     s1.print();
39     s2.print();
40     s3.print();
41     return 0;
42 }

```

```

g++ -g -o MyString1 MyString1.cpp
MyString1.cpp: In copy constructor 'MyString::MyString(const MyString&)':
MyString1.cpp:29:24: error: passing 'const MyString' as 'this' argument discards
qualifiers [-fpermissive]
    p = new char[s.size()+1]; // deep copy
                   ^
MyString1.cpp:13:7: note:   in call to 'int MyString::size()'
    int size(){ return strlen(p); }
    ^~~~~
Makefile:19: recipe for target 'MyString1' failed
make: *** [MyString1] Error 1

```

또 compiler error !


```

5  class MyString{
6      char *p;
7  public:
8      MyString(const char *str=NULL);
9      MyString(char *str=NULL);
10     MyString(const MyString& s);
11     ~MyString(){ delete[] p; }
12     void print() const { cout << p << endl; }
13     int size() const { return strlen(p); }
14 };
15 MyString::MyString(char *str){
16     cout << this << " " << str << "]" MyString(char *)\n";
17 }
18 MyString::MyString(const char *str){
19     if (!str){
20         p = new char[1];
21         p[0] = '\0';
22         return;
23     }
24     p = new char[strlen(str)+1];
25     strcpy(p, str);
26     cout << this << " " << str << "]" MyString(const char *)\n";
27 }
28 MyString::MyString(const MyString& s){
29     p = new char[s.size()+1]; // deep copy
30     strcpy(p, s.p);
31     cout << this << " " << s.p << "]" MyString(const MyString&)\n";
32 }

```

```

34 int main(){
35     const MyString s1("const");
36     MyString s2("C++");
37     MyString s3(s1);
38     s1.print();
39     s2.print();
40     s3.print();
41     return 0;
42 }

```

```

s1 0x7fff1d2c2780 const] MyString(const char *)
s2 0x7fff1d2c2788 C++] MyString(const char *)
s3 0x7fff1d2c2790 const] MyString(const MyString&)
s1 const
s2 C++
s3 const

```

```

4 class Kvector{
5     int *m;
6     int len;
7 public:
8     Kvector(int sz = 0, int value = 0);
9     Kvector(Kvector& v);
10    ~Kvector(){
11        cout << this << " : ~Kvector() \n";
12        delete[] m;
13    }
14    void print(){
15        for (int i=0; i<len; i++) cout << m[i] << " ";
16        cout << endl;
17    }
18    void clear(){
19        delete[] m;
20        m = NULL;
21        len = 0;
22    }
23    int size(){ return len; }
24 };
25 int main(){
26     Kvector v1(3); v1.print();
27     Kvector v2(2, 9); v2.print();
28     Kvector v3(v2); v3.print();
29     v2.clear();
30     v2.print();
31     v3.print();
32     return 0;
33 }

```



실습 : 지난 실습에서 구현한 Kvector class 에 대하여 main() 함수를 다음과 같이 변경하면 compile error 가 발생한다.

(1) compile error 가 생기지 않도록 class 멤버 함수들을 수정하라.

(2) 프로그램이 수행되는 동안 생성되는 Kvector 객체들에 저장된 m 배열의 원소 갯수들의 총합을 저장하는 static member 변수 total_len 을 선언하고 생성자와 소멸자를 수정하라.

```

38 int main(){
39     Kvector v1(3); v1.print();
40     const Kvector v2(2, 9); v2.print();
41     Kvector v3(v2); v3.print();
42
43     cout << "total length = " << Kvector::total_len << endl;
44     v2.print();
45     v3.print();
46     cout << "total length = " << Kvector::total_len << endl;
47     return 0;

```

```

38 int main(){
39     Kvector v1(3);  v1.print();
40     const Kvector v2(2, 9);  v2.print();
41     Kvector v3(v2);  v3.print();
42
43     cout << "total length = " << Kvector::total_len << endl;
44     v2.print();
45     v3.print();
46     cout << "total length = " << Kvector::total_len << endl;
47     return 0;

```

```

0x7ffd0f862660 : Kvector(int, int)
0 0 0
0x7ffd0f862670 : Kvector(int, int)
9 9
0x7ffd0f862680 : Kvector(Kvector&)
9 9
total length = 7
9 9
9 9
total length = 7
0x7ffd0f862680 : ~Kvector()
0x7ffd0f862670 : ~Kvector()
0x7ffd0f862660 : ~Kvector()

```


복소수 클래스

복소수: $a + bi$



```

4  class Complex{
5      double re, im;
6  public:
7      Complex(double r=0, double i=0);
8      ~Complex(){}
9      double real() {return re;}
10     double imag() {return im;}
11     Complex add(const Complex& c) const;
12     void print() const;
13 };
14
15 Complex::Complex(double r, double i): re(r), im(i){}
16 Complex Complex::add(const Complex& c) const{
17     Complex result(re + c.re, im + c.im);
18     return result;
19 }
20 void Complex::print() const{
21     cout << re << " + " << im << "i" <<endl;
22 }

```

```

23 int main(){
24     const Complex x(2,3);
25     Complex y(-1, -3), z;
26     x.print();
27     y.print();
28     z = x.add(y);
29     z.print();
30     return 0;
31 }

```

$2 + 3i$
 $-1 + -3i$
 $1 + 0i$