C++ Espresso

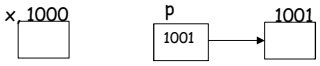
# 제7장 클래스의 활용


## 이번 장에서 학습할 내용

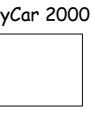
- 객체의 동적 생성
- this
- const
- 객체와 연산자
- 객체와 함수
- 정적 멤버
- 객체의 배열(앞으로 이동)

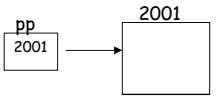
# 객체의 동적 생성(객체 포인터)

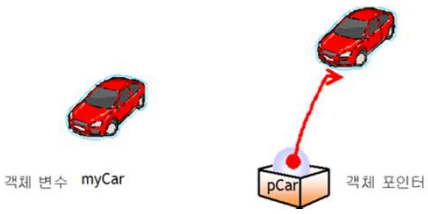
- 객체도 동적으로 생성할 수 있다.




```
int x, *p;
p = new int;            p = &x;

///////////////
Car   myCar, *pp;
pp = new Car( );        // 동적 메모리 할당으로 객체 생성 → 디폴트 생성자 호출 명시
// pp = new Car;        // 이 방법도 가능하지만 위 방법이 일반적.

pp->speed = 200;
pp = &myCar;

Car *pCar = new Car( );          // 디폴트 생성자 호출
pCar->speed = 100;
pCar->speedUp();
```



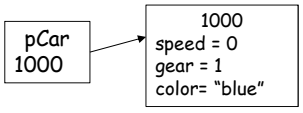




# 예제

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

class Car {
      int speed, gear;
      string color;
 public:
   Car(int s=0, int g=1, string c="white")
        : speed(s), gear(g), color(c) {  }

   void print()
   {
      cout << "속도: " << speed << " 기어: " <<
              gear << " 색상: " << color << endl;
   }
};
```

```
int main() {
      Car myCar, *pCar;

      myCar.print();                    객체 동적 생성

      pCar = new Car(0, 1, "blue");
      pCar->print();

      delete pCar;  // 책에 없음.

      return 0;
}
```

```
속도: 0 기어: 1 색상: white
속도: 0 기어: 1 색상: blue
```




# 객체들의 배열

- 참고) 교재 순서상 뒷 부분이지만 여기서 수업
- int x[3];
- Car objArray[3];

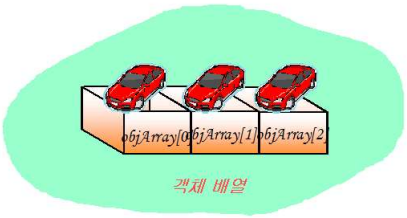


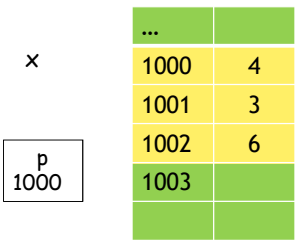

그림 10.8 객체 배열

objArray[0].speed = 0;// 멤버 변수 접근

objArray[1].speedUp();// 멤버 함수 호출

# C 에서 배열과 포인터

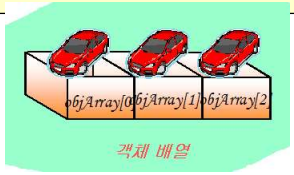
```
int main() {
        int x[3] = {4, 3, 6};  // 배열 초기화
        int *p;

        for(int i=0; i< 3; i++)
                cout << x[i];

        p = x;     // 배열 이름은 배열 시작 주소
        for(int i=0; i< 3; i++)
                cout << *(p+i);


        for(int i=0; i< 3; i++)
                cout << p[i];

        return 0;
}
```




# 예 제

배열 초기화시 마지막 초기값 다음에 " , " 있거나 없거나 무관

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

class Car {
   int speed, gear;
   string color;

public:
   Car(int s=0, int g=1, string c="white")
      : speed(s), gear(g), color(c) {    }

   void print() {
      cout << "속도: " << speed << " 기어: "
           << gear << " 색상: " << color
           << endl;
   }
};
```



객체 배열

```
int main() {
     // 객체 배열 초기화, 객체 별로 생성자 호출

     Car objArray[3] = {
          Car(0, 1, "white"),
          Car(0, 1, "red"),
          Car(0, 1, "blue") ,
     };

     for(int i=0; i< 3; i++)
          objArray[i].print();

     return 0;
}
```

```
속도: 0 기어: 1 색상: white
속도: 0 기어: 1 색상: red
속도: 0 기어: 1 색상: blue
계속하려면 아무 키나 누르십시오 . . .
```

# 예 제

```
int main(){
          Car    objArray[3] = { Car(1, 2, "red"), Car() };
          for(int i=0; i< 3; i++)
                    objArray[i].print();

          return 0;
}
```

```
속도: 1 기어: 2 색상: red
속도: 0 기어: 1 색상: white
속도: 0 기어: 1 색상: white
```

- 배열 생성시 생성자 호출
  - 1st : 변수값 주고 호출
  - 2nd : 변수값 없으니 매개변수 초기화값 사용
  - 3rd : 코드에는 객체 생성 내용 없으나 배열의 3번째 방(객체)가 생성되기에 매개변수 없이 생성자 호출

```
//앞 페이지 생성자
Car(int s=0, int g=1, string c="white")
     : speed(s), gear(g), color(c) {    }
```

# 객체 배열과 포인터

- 객체 배열의 이름은 포인터처럼 사용될 수 있다.

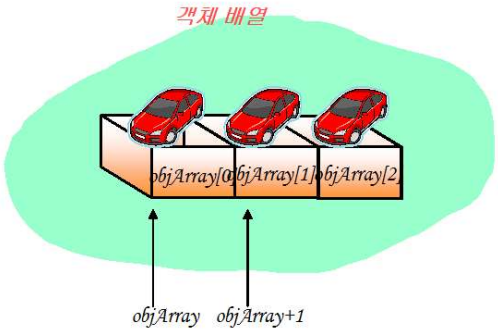



그림 7.8 객체의 이름은 포인터

# 예제

```
int main(){
        Car     objArray[3] = { Car(0, 1, "white"), Car(0, 1, "red"), Car(0, 1, "blue"),};
        for(int i=0; i< 3; i++)
                objArray[i].print();

        for(int i=0; i< 3; i++)                 // 배열 이름은 배열 시작 주소
                (objArray+i)->print();

        Car *p = objArray;
        for(int i=0; i< 3; i++){
                p->print();
                p++;
        }
        p = objArray;                   // 배열의 맨 앞을 다시 지정
        for(int i=0; i< 3; i++)   // 포인터를 배열같이 사용
                p[i].print();

        return 0;
}
```

objArray

p
1000

| ... | |
|---|---|
| 1000 | 0, 1, w |
| 1001 | 0, 1, r |
| 1002 | 0, 1 , b |
| 1003 | |
| | |

```
속도: 0 기어: 1 색상: white
속도: 0 기어: 1 색상: red
속도: 0 기어: 1 색상: blue
속도: 0 기어: 1 색상: white
속도: 0 기어: 1 색상: red
속도: 0 기어: 1 색상: blue
```

# this 포인터

- this는 멤버함수를 호출한 객체를 지칭 → 해당 멤버 변수/함수를 소유하는 객체를 지칭

```
//Car 클래스는 앞과 동일

void Car::setSpeed(int speed){
   speed = speed;     // speed 동일 ??
}                     // Error
```

⇨

```
void Car::setSpeed(int speed){
   this->speed = speed;
}
```

```
void main(){
   Car x, y;
   x.setSpeed(100);
   y.setSpeed(10);
}
```

x

```
speed
void setSpeed(int speed){
   this->speed = speed;
}
```

y

```
speed
void setSpeed(int speed){
   this->speed = speed;
}
```

# this 사용하지 않은 경우(없어도 무방, 코드 혼란)

- this는 멤버함수를 호출한 객체, 해당 멤버를 소유하는 객체

```
class Car {
   int speed, gear;
   string color;
public:
   Car(int s=0, int g=1, string c="white")
         : speed(s), gear(g), color(c) {   }

   void isFaster(Car *p);
   int getSpeed( );
   void print() ;
};
```

```
int main()
{
      Car c1(0, 1, "blue");
      Car c2(100, 3, "red");
      c1.isFaster(&c2);

      return 0;
}
```

```
void Car::isFaster(Car *p){   // Car *p = &c2;
   if( getSpeed() > p->getSpeed() )
            print();
   else
            p->print();
   cout << "의 자동차가 더 빠름" << endl;
}
```

```
void Car::print()   {
   cout << "속도: " << speed << " 기어: " <<
            gear << " 색상: " << color << endl;
}
int Car::getSpeed( ) {
   return speed;
}
```

## this 사용한 경우(결과는 앞 코드와 동일, 명쾌)

- this는 멤버함수를 호출한 객체, 해당 멤버함수를 소유하는 객체

```
c1
speed=0, gear=1,
color = "blue"
Car(… )
getSpeed( )
print( )
isFaster( ) { this…}
```

```
c2
speed=100, gear=3,
color="red"
Car( …)
getSpeed( )
print( )
isFaster( ){ this…}
```

```
int main()
{
        Car c1(0, 1, "blue");
        Car c2(100, 3, "red");
        c1.isFaster(&c2);
        return 0;
}
```

```
void Car::isFaster(Car *p){ // Car *p=&c2;
   if( this->getSpeed() > p->getSpeed() )
          this->print();
   else
          p->print();
   cout << "의 자동차가 더 빠름" << endl;
}
```

```
void Car::print()  {
   cout << "속도: " << speed << " 기어: " <<
          gear << " 색상: " << color << endl;
}
int Car::getSpeed( ) {
   return this->speed; // 사용 불필요
}
```

참고) *C#* 에서는 this.xxx,  *C++* 에서는 this->xxx, this 는 포인터

## this 포인터 사용 예

- this는 멤버함수를 호출한 객체, 해당 멤버를 소유하는 객체

```
void Car::setSpeed(int speed)
{
    if( speed > 0 )                  // 다음과 같이 변수 혼동 있는 경우 this 사용
        this->speed = speed;         // speed는 매개 변수, this->speed는 멤버 변수
    else                             // speed = speed; 하면 error
        this->speed = 0;             // 멤버변수, 인자 구분 불가 → 꼭 사용
}
```

```
// 생성자
Car::Car(int s) {
    this->setSpeed(s);     // 혼동 없는 경우, this는 없어도 된다.
    this->gear = 1;
    this->color = "white";
}
```

- 멤버함수 호출시 this 는 매개변수로 전달됨
- 함수 호출한 객체를 지칭

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

class Car {
   int speed; // 속도
   int gear; // 기어
   string color; // 색상

public:
   Car(int s=0, int g=1, string c="white")
       : speed(s), gear(g), color(c) { }

   int getSpeed() {
              return speed;
   }
   void setSpeed(int speed) {
      if( speed > 0 )
          this->speed = speed;
      else
          this->speed = 0;
   }
   void print() {
      cout << "속도: " << speed << " 기어: "
           << gear << " 색상: " << color;
   }

   void isFaster(Car *p) {
      if( this->getSpeed() > p->getSpeed() )
         this->print();
      else
         p->print();
      cout << "의 자동차가 더 빠름" << endl;
   }
};
```

```
int main()
{
              Car c1(0, 1, "blue");
              Car c2(100, 3, "red");
              c1.isFaster(&c2);
              return 0;
}
```

속도: 100 기어: 3 색상: red의 자동차가 더 빠름

# const 수식어

- 멤버 변수에 **const**를 붙이는 경우

```
class Car
{
              const int serial;
              string color;
              ...
public:
              Car(int s, string c) : serial(s) {       // 상수멤버 초기화 : 6장
                            color = c;
              }
}
```

이 멤버 변수의 값을 변경할 수 없다.

- 멤버 함수 뒤에 **const**를 붙이는 경우

```
void displayInfo() const
{
              cout << "속도: " << speed << endl;
              cout << "기어: " << gear << endl;
              cout << "색상: " << color << endl;
}
```

이 함수 안에서는 멤버 변수의 값을 변경할 수 없다.

# const 수식어

- 어떤 멤버함수를 const 로 작성하나 ?
  - 출력용 함수, 멤버변수값 알아보는 함수들 → 멤버변수를 변경하지 않는 멤버함수들
  - 혹시 멤버변수 변경하지 않아야 하는데 실수로 변경하는 것 방지
- const 함수 내에서는 const 함수만 호출 가능
  - const 아닌 함수 호출하여 그 함수에서 멤버변수 변화 가능성 방지

```
int Car::getSpeed( ){
   return speed;
}

void Car::displayInfo() const  {
            cout << getSpeed( ) << endl;     // Error
}
```

```
int Car::getSpeed( ) const {
   return speed;
}

void Car::displayInfo() const  {
            cout << getSpeed( ) << endl;     // OK
}
```

# const 수식어

- 객체에 const를 붙이는 경우

```
int main(){
            const Car c1(0, 1, "yellow");
            c1.setSpeed(100);  // 오류!, c1 객체 멤버변수 변경 금지
            return 0;
}

void Car::setSpeed(int speed) {
    if( speed > 0 )    this->speed = speed;
    else               this->speed = 0;
}
```

이 객체의 멤버 변수 값을 변경할 수 없다.

# const 수식어

- 함수 매개변수의 객체에 const를 붙이는 경우(많이 사용)
  - 해당 객체의 멤버변수 값 변경 불가, const 객체는 const 함수만 호출 가능

```
int main(){
   Car c1(0, 1, "yellow"), c2(100, 2, "red");
   sub(c1, c2);              return 0;
}
                                        // "const Car &cc = c2" 로 해석
void sub(Car& c, const Car& cc){ // 매개변수 const 사용한 경우
   int s1 = c.getSpeed();
   c.setSpeed(100);                     // OK

   int s2 = cc.getSpeed();
   cc.setSpeed(100);                    // error, cc 는 const
}

void Car::setSpeed(int speed) {     // const 함수 아님
     if( speed > 0 )       this->speed = speed;
     else                  this->speed = 0;
}
```

# 객체와 연산자

string 클래스에서는 >, == 등 연산자 사용 가능 → 기존에 만들어져 있음

- 객체에 할당 연산자(=), 비교연산자(==) 를 사용할 수 있는가?

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

        if( c1 == c2 )
                cout << "같습니다" << endl;
        else
                cout << "같지않습니다" << endl;
        return 0;
}
```

c2 객체가 가지고 있는 변수의 값이 c1으로 복사된다..

연산자 정의 되어 있지 않으면 오류!
== 함수 없으면 오류

- 개발자가 만든 클래스의 객체 사용시
- 가장 기본적인 연산자인 " = " (대입연산자)는 정의 되어 있으나
- ==, > 등 의 다른 연산자들은 없음 → 작성하여 사용해야 함.(10장)

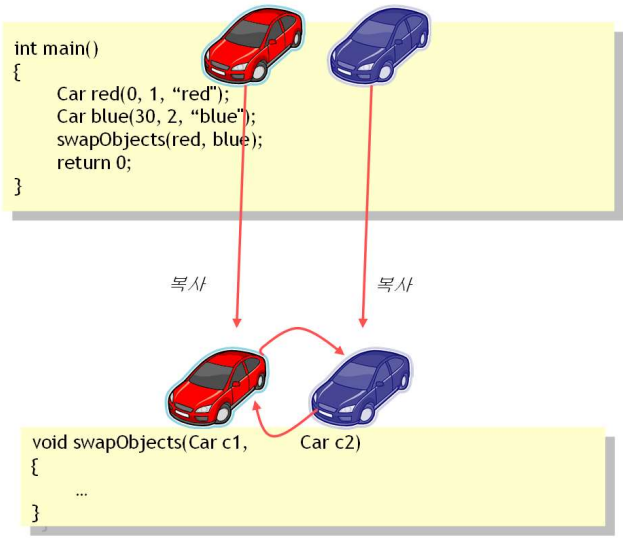
# 객체와 함수(3장 설명 동일 내용)

① 객체가 함수의 매개 변수로 전달되는 경우
② 함수가 객체를 반환하는 경우
③ 객체의 포인터가 함수의 매개 변수로 전달되는 경우
④ 객체의 레퍼런스(참조자)가 함수의 매개 변수로 전달되는 경우

```
int a, b;

a = 20;   b = 30;

swap_1(a, b);
cout << a << b;

swap_2(&a, &b);
cout << a << b;

swap_3(a, b);
cout << a << b;
```

```
void swap_1(int x, int y){
  int t;
  t = x;   x = y;      y = t;
}

void swap_2(int *x, int *y){
  int t;
  t = *x;   *x = *y;      *y = t;
}

void swap_3(int &x, int &y){
  int t;
  t = x;   x = y;      y = t;
}
```

# 1. 객체가 함수의 매개 변수로 전달



# 객체가 함수의 매개 변수로 전달

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

class Car {
  int speed;
  int gear;
  string color;

public:
  Car(int s=0, int g=1, string c="white")
    : speed(s), gear(g), color(c) {     }

  void print()  {
     cout << "속도: " << speed << " 기어: " <<
         gear << " 색상: " << color << endl;
  }
};
```

```
// 일반 함수
            // Car c1 = red;     Car c2 = blue;
void swapObjects(Car c1, Car c2) {
            Car tmp;
            tmp = c1;                  c1 = c2;
            c2 = tmp;
} // 반환값 없음.

int main() {
            Car red(0, 1, "red");
            Car blue(30, 2, "blue");

            swapObjects(red, blue);
            red.print();
            blue.print();
            return 0;
}
```

```
속도: 0 기어: 1 색상: red
속도: 30 기어: 2 색상: blue
```

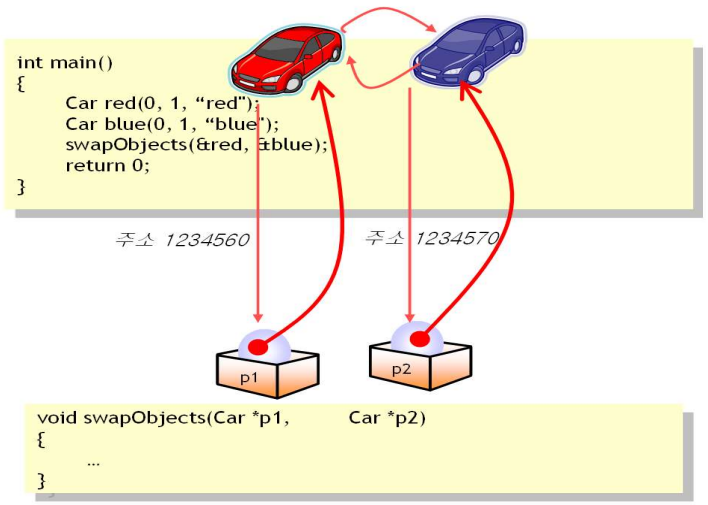
# 2. 함수가 객체를 반환

```
...// Car 클래스 전과 동일

Car createCar() {       // 일반함수
            Car tmp(0, 1, "metal");
            return tmp;
}

int main(){
            Car c;                  // 생성자 인자 초기값 사용
            c.print();

            c = createCar();        // 멤버함수 아님
            c.print();

            return 0;
}
```

함수 반환형과 반환값은 동일해야 함

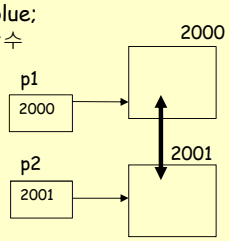
```
속도: 0 기어: 1 색상: white
속도: 0 기어: 1 색상: metal
```

# 3. 객체의 포인터가 함수에 전달




# 3. 객체의 포인터가 함수에 전달

```
...// Car 클래스 전과 동일
                    Car *p1=&red;    Car *p2=&blue;
void swapObjects(Car *p1, Car *p2) {  // 일반함수
        Car tmp;

        tmp = *p1;
        *p1 = *p2;
        *p2 = tmp;
}

int main() {
        Car red(0, 1, "red");
        Car blue(0, 1, "blue");

        swapObjects(&red, &blue);
        red.print();
        blue.print();

        return 0;
}
```




```
속도: 0 기어: 1 색상: blue
속도: 0 기어: 1 색상: red
```

# 4. 객체의 참조자가 함수에 전달

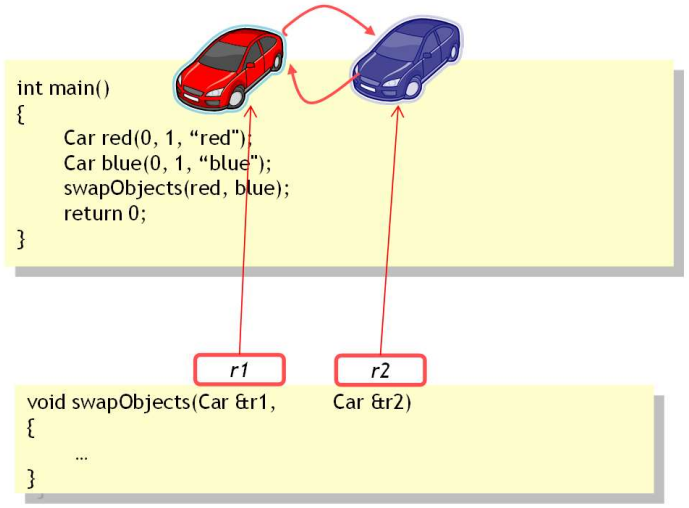
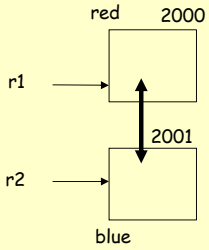



그림 7.5 객체의 참조자를 전달하는 경우

# 4. 객체의 참조자가 함수에 전달

```
...// Car 클래스 전과 동일
                    Car &r1=red;    Car &r2=blue;
void swapObjects(Car &r1, Car &r2){
        Car tmp;

        tmp = r1;
        r1 = r2;
        r2 = tmp;
}
int main(){
        Car red(0, 1, "red");
        Car blue(30, 2, "blue");

        swapObjects(red, blue);
        red.print();
        blue.print();
        return 0;
}
```

```
속도: 0 기어: 1 색상: blue
속도: 0 기어: 1 색상: red
계속하려면 아무 키나 누르십시오 . . .
```




# 임시 객체 (6장 설명)

- 함수가 객체를 반환하는 경우에도 임시 객체 생성

*temp_obj.cpp*

```
class Car {
...
};
Car createCar()
{
  Car tmp(0, 1, "metal");
  return tmp;
}

int main()
{
  createCar().print();
  return 0;
}
```

임시 객체

함수를 호출하여
반환된 객체 안의 멤버함수 호출

반환된 임시객체를 통하여 멤버 함수 호출

임시객체.print( )

# 임시 객체(이름 없는 객체)

- 참고) string::c_str( ) → sting 객체가 가지고 있는 문자열 시작주소 반환
  - 참고) string 을 char * 로 형변형 시 사용(형 변환, printf 사용 출력시)
- 수식의 계산 도중에 중간 결과를 임시로 저장하기 위하여 임시적으로 만들어지는 객체 → 임시객체는 그 문장 수행 후 바로 소멸

```
int main()
{
  string s1 = "Hello ";
  string s2 = "World";
  const char* p = (s1+s2).c_str();      // ①
  cout << p;                            // 아무 출력 없음

  return 0;
}
```

임시 객체가 생성된다.

  - 임시객체 생성(tmp, 2000 번지, “Hello world” 저장) → p 는 2000번지 저장
  - 다음 줄로 가면 tmp 객체 소멸(내용 소멸) → 2000 번지 내용은 없음
- 출력 되려면 ?

```
string s3 = s1 + s2;
const char* p1 = s3.c_str();
```

# 임시 객체

```
int main( ) {
        string s1 = "Hello ", s2 = "World", s3 = s1+s2, s4;

        const char *p = (s1+s2).c_str();  // 임시 객체
        cout << p;

        const char *p1 = s3.c_str();
        cout << p1;

        return 0;
}
```

```
출력 없음

Hello World

계속하려면 아무 키나 누르십시오 . . .
```