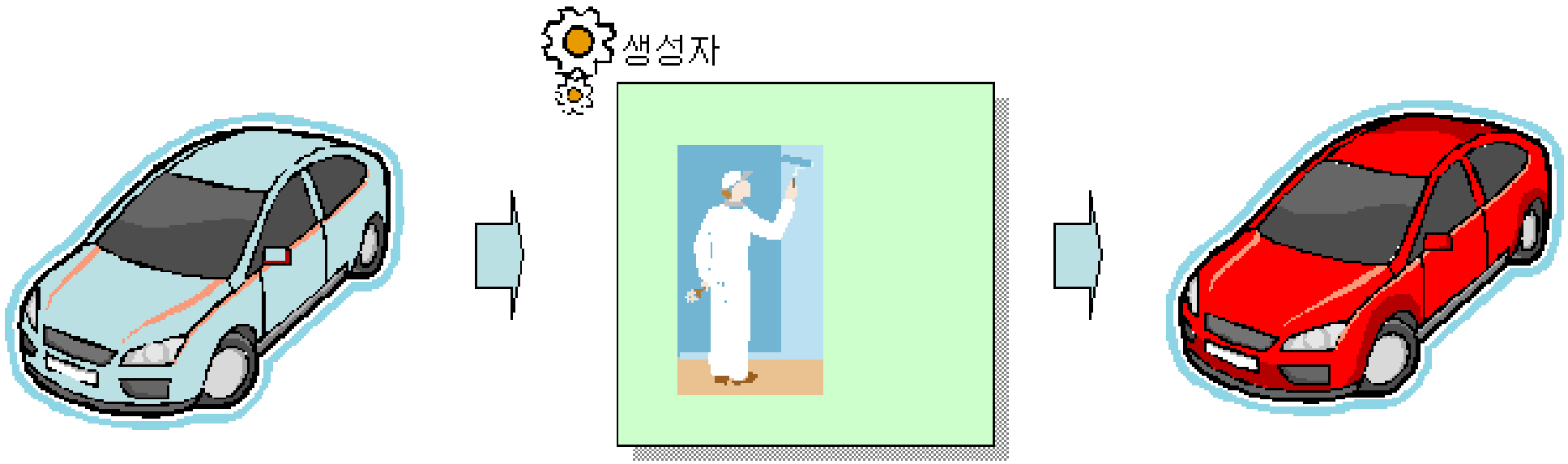
생성자와 소멸자
Constructors and Destructors

2023

국민대학교 소프트웨어학부

생성자

- 생성자(constructor): 객체가 생성될 때에 필드에게 초기값을 제공하고 필요한 초기화 절차를 실행하는 멤버 함수



객체의 일생

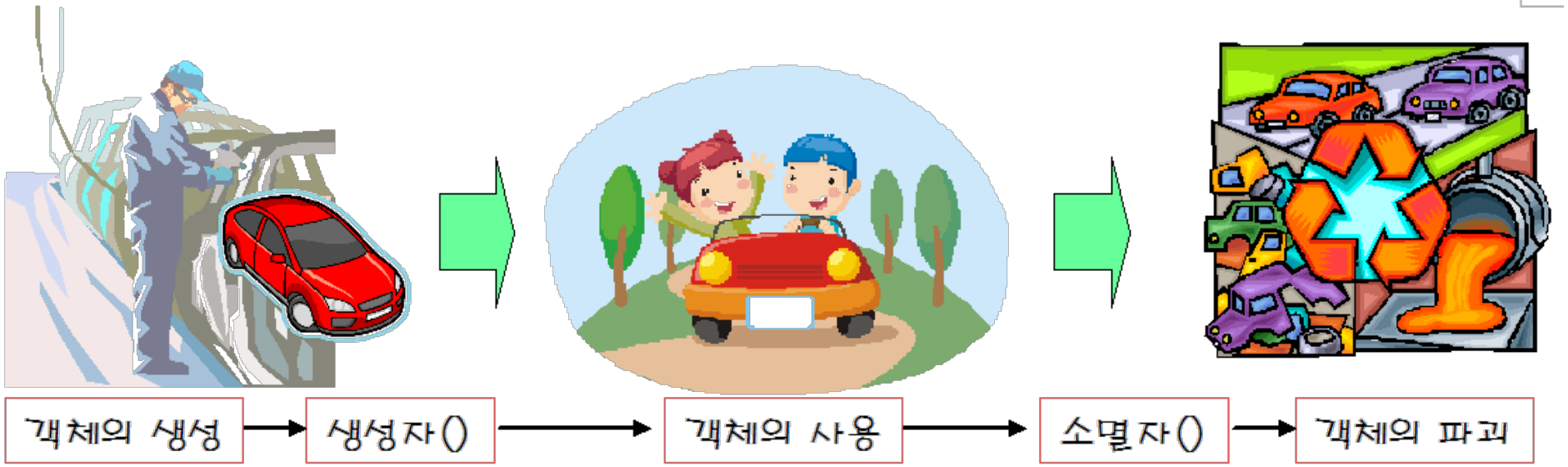
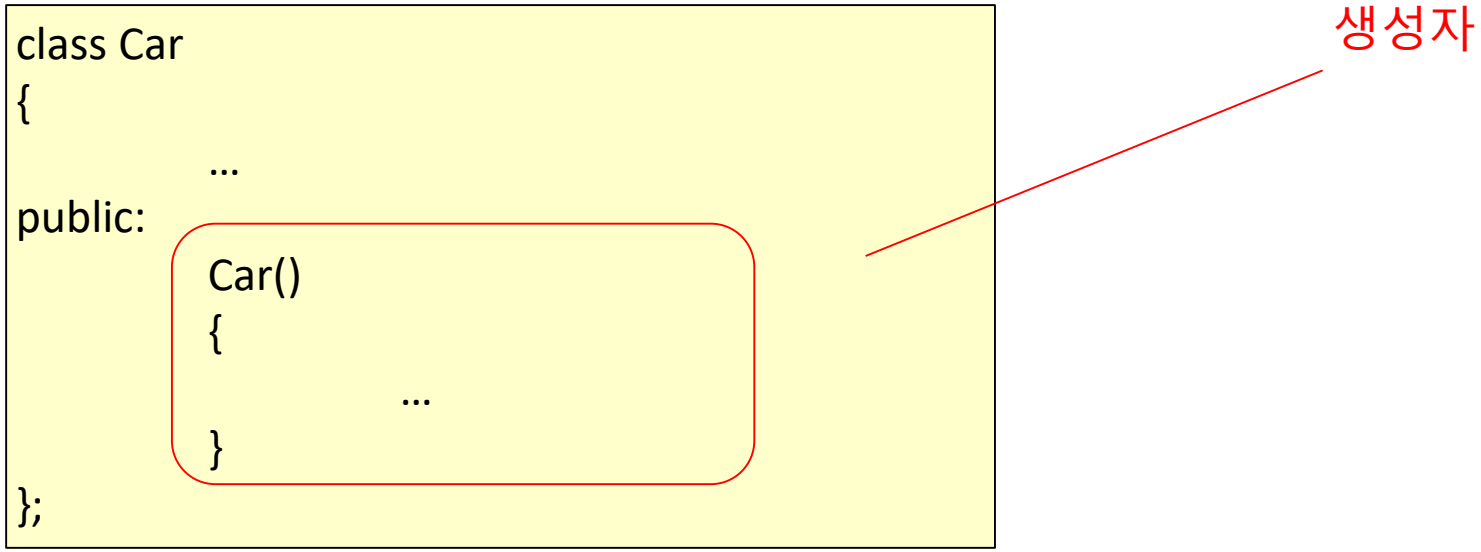


그림 10.2 객체의 일생

생성자의 특징

- 클래스 이름과 동일하다
- 반환값이 없다.
- 반드시 public 이어야 한다.
- 중복 정의할 수 있다.



Default constructor

- parameter 가 없는 생성자. (혹은 parameter가 모두 default 값을 갖는 생성자)
- 클래스의 생성자가 아무 것도 없으면 compiler 가 자동으로 만든다.

생성자를 하나도 정의하지 않으면?

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

컴파일러가 비어있는 디폴트 생성자를 자동으로 추가한다.

```
class Car {
      int speed;          // 속도
      int gear;           // 기어
      string color;       // 색상
public:
      Car() { }
}
```

```
class A{
  int p;
public:
  A(){
     cout << "A() called\n";
     p = 1;
  }
  A(int v){
     cout << "A(int) called\n";
     p = v;
  }
  int getP(){
     return p;
  }
  void setP(int v){
     p = (v<0)? 0 : v;
  }
};
int main(){
  A a1, a2(10);
  cout << a1.getP() << endl;
  cout << a2.getP() << endl;
  return 0;
}
```

```
A() called
A(int) called
1
10
```

```
4  class A{
5    int p;
6  public:
7  /*
8    A(){
9       cout << "A() called\n";
10      p = 1;
11   }
12 */
13   A(int v){
14      cout << "A(int) called\n";
15      p = v;
16   }
17   int getP(){
18      return p;
19   }
20   void setP(int v){
21      p = (v<0)? 0 : v;
22   }
23 };
24
25 int main(){
26   A a1, a2(10);
27   cout << a1.getP() << endl;
28   cout << a2.getP() << endl;
29   return 0;
30 }
```

생성자가 하나라도 있으면default 생성자는 자동 생성되지 않는다.

```
ejim@ejim-VirtualBox:~/C2020$ make constructor
g++ -g -o constructor constructor.cpp
constructor.cpp: In function ‘int main()’:
constructor.cpp:26:5: error: no matching function for call to ‘A::A()’
   A a1, a2(10);
     ^~
constructor.cpp:13:3: note: candidate: A::A(int)
   A(int v){
   ^
constructor.cpp:13:3: note:   candidate expects 1 argument, 0 provided
constructor.cpp:4:7: note: candidate: constexpr A::A(const A&)
 class A{
       ^
constructor.cpp:4:7: note:   candidate expects 1 argument, 0 provided
constructor.cpp:4:7: note: candidate: constexpr A::A(A&&)
constructor.cpp:4:7: note:   candidate expects 1 argument, 0 provided
Makefile:19: recipe for target 'constructor' failed
make: *** [constructor] Error 1
```

```
4  class A{
5    int p;
6  public:
7  /*
8    A(){
9       cout << "A() called\n";
10      p = 1;
11   }
12 */
13   A(int v = 0){
14      cout << "A(int) called\n";
15      p = v;
16   }
17   int getP(){
18      return p;
19   }
20   void setP(int v){
21      p = (v<0)? 0 : v;
22   }
23 };
24
25 int main(){
26   A a1, a2(10);
27   cout << a1.getP() << endl;
28   cout << a2.getP() << endl;
```

default parameter 가 있기 때문에 default constructor 가 된다.

```
A(int) called
A(int) called
0
10
```

```
4  class A{
5    int p;
6  public:
7  /*
8    A(){
9       cout << "A() called\n";
10      p = 1;
11   }
12   A(int v = 0){
13      cout << "A(int) called\n";
14      p = v;
15   }
16 */
17   int getP(){
18      return p;
19   }
20   void setP(int v){
21      p = (v<0)? 0 : v;
22   }
23 };
24 A a2;
25 int main(){
26   A a1;
27   cout << a1.getP() << endl;
28   cout << a2.getP() << endl;
29   return 0;
30 }
```

⇨ default constructor A::A(){} 가 자동으로 생성됨

```
32767
0
```

default constructor 는 아무 것도 하지 않기 때문에
지역 변수는 초기화되지 않고
전역 변수는 0으로 초기화된다.

생성자 호출의 다양한 방법

```
int main()
{
        Car c1;     // ①디폴트 생성자 호출
        Car c2();  // ②이것은 생성자 호출이 아니라 c2()라는 함수의 원형 선언
        Car c3(100, 3, "white");        // ③생성자 호출
        Car c4 = Car(0, 1, "blue");// ④이것은 먼저 임시 객체를 만들고 이것을
                                   c4에 복사    same as Car c4(0,1,"blue")
        return 0;
}
```

소멸자

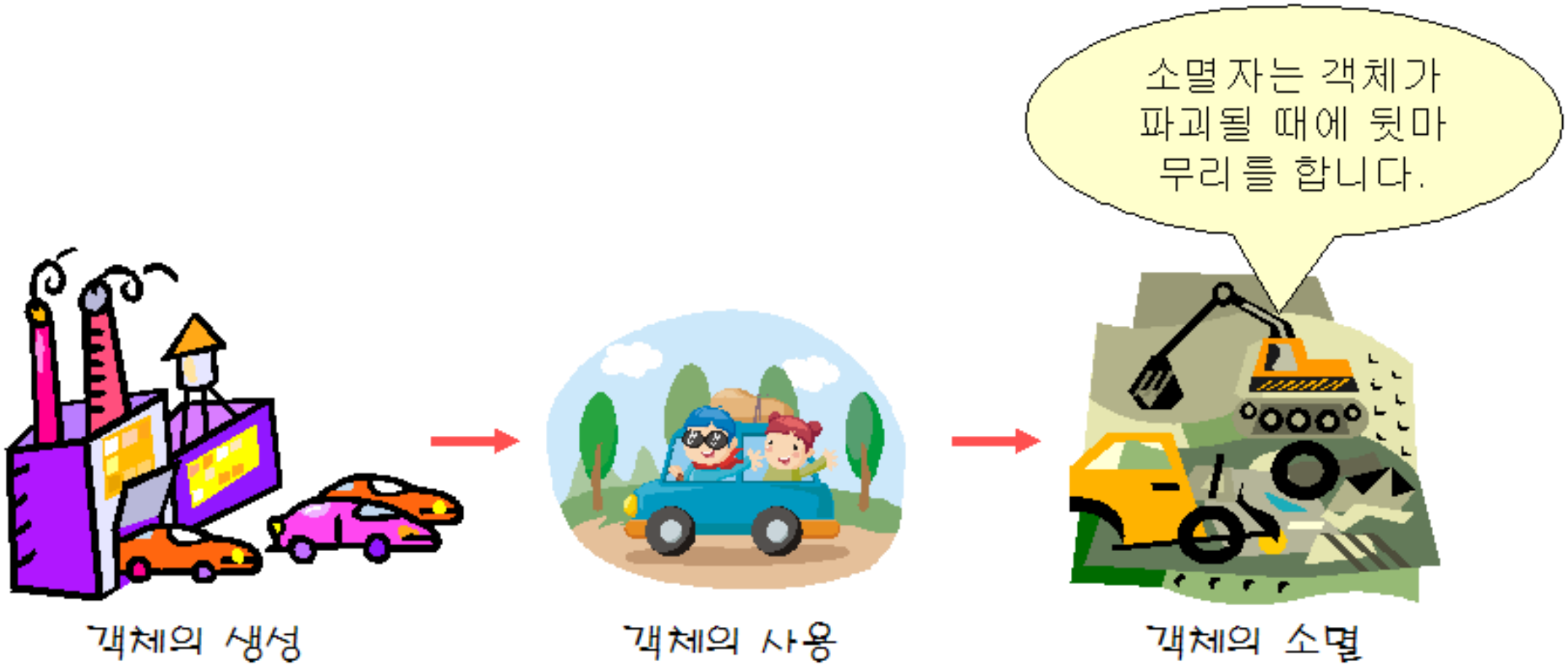
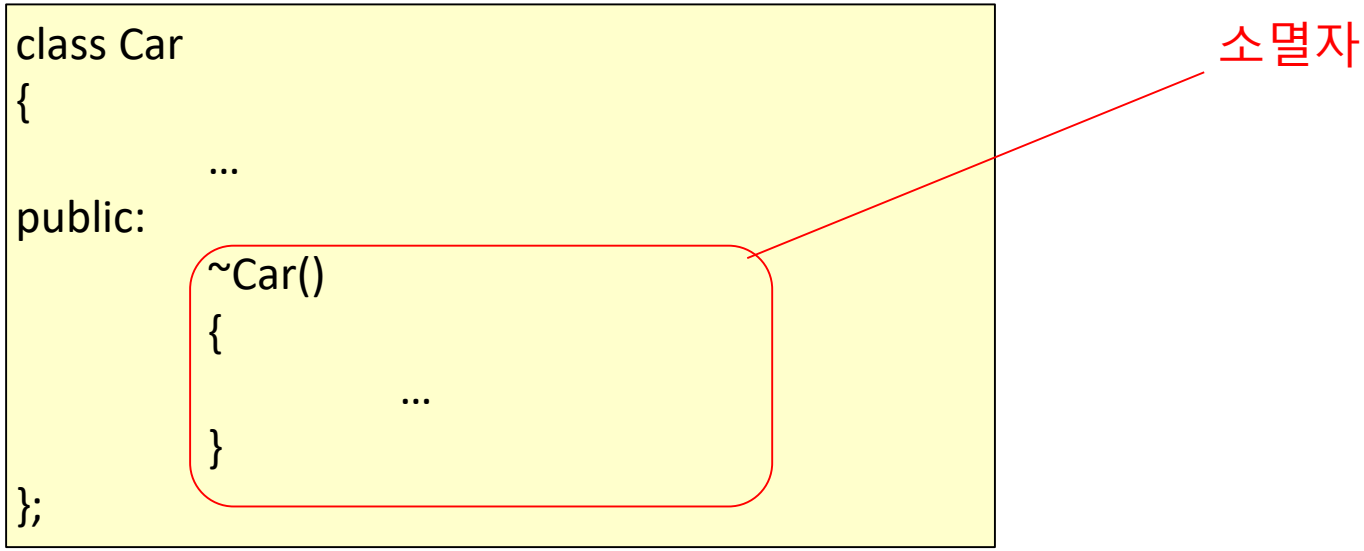


그림 10.4 소멸자의 개념

소멸자의 특징

- 소멸자는 클래스 이름에 ~가 붙는다.
- 값을 반환하지 않는다.
- public 멤버 함수로 선언된다.
- 소멸자는 매개 변수를 받지 않는다.
- 중복 정의도 불가능하다.



```
4  class A{
5    int p;
6  public:
7    A(int v = 0){
8       cout << "A(int) called\n";
9       p = v;
10   }
11   ~A(){
12     cout << "~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 }
```

```
A(int) called
A(int) called
0
2
~A() called
~A() called
```

which one?

```
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
```

디폴트 소멸자

- 만약 프로그래머가 소멸자를 정의하지 않았다면 어떻게 되는가?
- 디폴트 소멸자가 자동으로 삽입되어서 호출된다

```
class Time {
        int hour, minute, second;
public:
}
```

~Time() { } 을 넣어준다.

```
4  class Kvector{
5    int *m;
6  public:
7    Kvector(int sz = 0){
8       cout << this << " : Kvector(int) called\n";
9       if (sz > 0) m = new int[sz];
10      else        m = NULL;
11   }
12   ~Kvector(){
13     cout << this << " : ~Kvector() called\n";
14     delete[]  m;
15   }
16   int *getM(){
17      return m;
18   }
19 };
20 Kvector v2(2);
21 int main(){
22   Kvector v1;
23   cout << v1.getM() << endl;
24   cout << v2.getM() << endl;
25   return 0;
26 }
```

동적 할당 받은 메모리를
반납하는 소멸자는
메모리 누수를 방지하기
위해 반드시 필요

```
0x563512ddb138 : Kvector(int) called
0x7ffea28b13a0 : Kvector(int) called
0
0x563513070280
0x7ffea28b13a0 : ~Kvector() called
0x563512ddb138 : ~Kvector() called
```

초기화 리스트

- 생성자에서 멤버 변수를 간단히 초기화할 수 있는 형식

```
4  class A{
5    int p;
6  public:
7    A(int v = 0){
8       cout << this << " : A(int) called\n";
9       p = v;
10   }
```

=

```
4  class A{
5    int p;
6  public:
7    A(int v = 0): p(v){
8       cout << this << " : A(int) called\n";
9    }
```

선언과 동시에 초기화되어야 하는 변수(상수)들은 초기화 리스트로만 초기화할 수 있다.

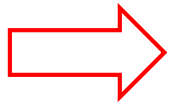
- symbolic constant
- reference

초기화 리스트를 사용해야 하는 경우 1) 상수 멤버

```
class Car
{
        const int MAX_SPEED = 300;// 컴파일 오류
                                  //(C++11 부터 허용)
        int speed;
        ...
}
```

아직 생성이 안됐음!

```
class Car
{                                          상수를 변경할 수 없음!
        const int MAX_SPEED;
        int speed;
public:
        Car()
        {
                MAX_SPEED = 300; // 컴파일 오류!
        }
}
```



```
class Car
{
        const int MAX_SPEED;

        int speed;
public:
        Car() : MAX_SPEED(300)
        {
        }
};
```

```
4  class A{
5    int p;
6    const int c;
7  public:
8    A(int v = 0, int c = 0) : p(v), c(c){
9       cout << this << " : A(int, int) called\n";
10   }
11   ~A(){
12     cout << this << " : ~A() called\n";
13   }
14   int getP(){
15      return p;
16   }
17   int getC(){
18      return c;
19   }
20   void setP(int v){
21      p = (v<0)? 0 : v;
22   }
23 };
24 A a2(2,3);
25 int main(){
26   A a1(10);
27   cout << a1.getP() << endl;
28   cout << a1.getC() << endl;
29   cout << a2.getP() << endl;
30   cout << a2.getC() << endl;
```

이름이 같아도 된다.

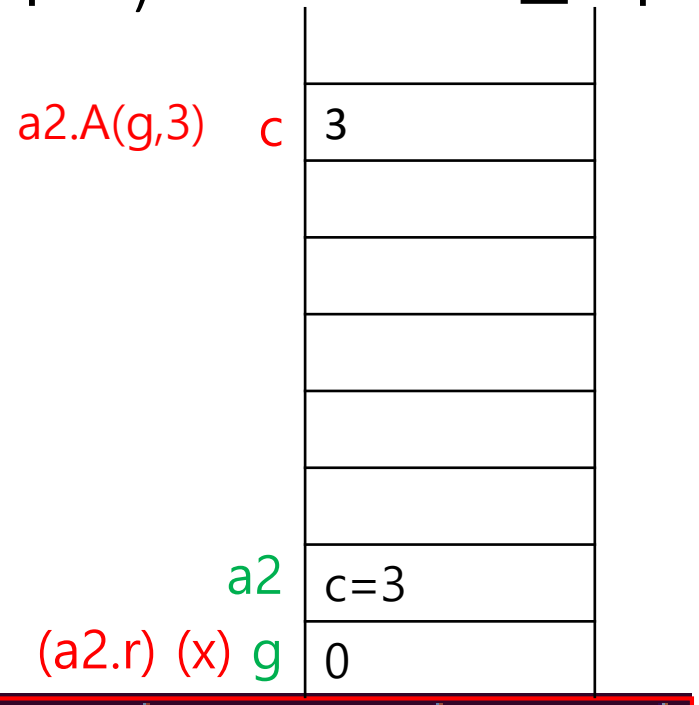
상수 멤버의 초기화: instance 마다 상수의 값이 다르게 할 수 있음

```
0x55cf98844138 : A(int, int) called
0x7ffdb3548a30 : A(int, int) called
10
0
2
3
0x7ffdb3548a30 : ~A() called
0x55cf98844138 : ~A() called
```

초기화 리스트를 사용해야 하는 경우 2) reference 멤버

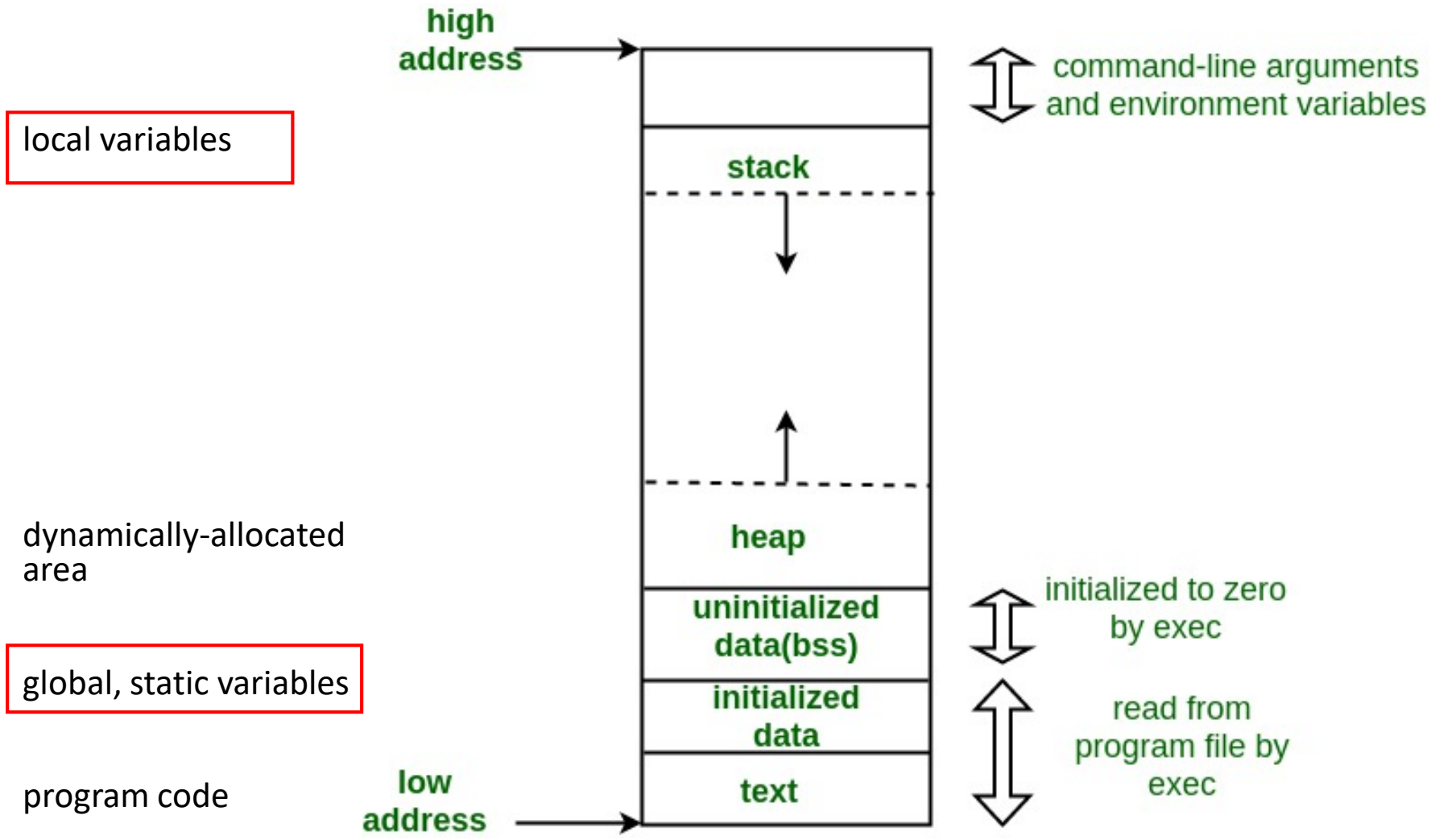
```
4  class A{
5    const int c;
6    int &r;
7  public:
8    A(int &x, int c = 0) : c(c), r(x){
9        cout << this << " : A(int&, int) called\n";
10   }
11   ~A(){ cout << this << " : ~A() called\n";}
12   int getR(){     return r;}
13   int getC(){     return c;}
14   void setR(int v){ r = v; }
15 };
16 int g;
17 A a2(g, 3);
18 int main(){
19   int i = 100;
20   A a1(i, 10);
21   cout << a1.getR() << endl; a1.setR(-1);
22   cout << a1.getR() << " i : " << i << endl;
23   cout << a1.getC() << endl;
24   cout << a2.getR() << endl; a2.setR(-2);
25   cout << a2.getR() << " g : " << g << endl;
26   cout << a2.getC() << endl;
27   return 0;
```

int &x = g;
int &r = x;




```
0x55a9be7af150 : A(int&, int) called
0x7fffc12f8c80 : A(int&, int) called
100
-1 i : -1
10
0
-2 g : -2
3
0x7fffc12f8c80 : ~A() called
0x55a9be7af150 : ~A() called
```

Typical Memory Layout of a C program



```
4  class A{
5    const int c;
6    int &r;
7  public:
8    A(int &x, int c = 0) : c(c), r(x){
9        cout << this << " : A(int&, int) called\n";
10   }
11   ~A(){ cout << this << " : ~A() called\n";}
12   int getR(){     return r;}
13   int getC(){     return c;}
14   void setR(int v){ r = v; }
15 };
16 int g;
17 A a2(g, 3);
18 int main(){
19   int i = 100;
20   A a1(i, 10);
21   cout << a1.getR() << endl; a1.setR(-1);
22   cout << a1.getR() << " i : " << i << endl;
23   cout << a1.getC() << endl;
24   cout << a2.getR() << endl; a2.setR(-2);
25   cout << a2.getR() << " g : " << g << endl;
26   cout << a2.getC() << endl;
27   return 0;
```

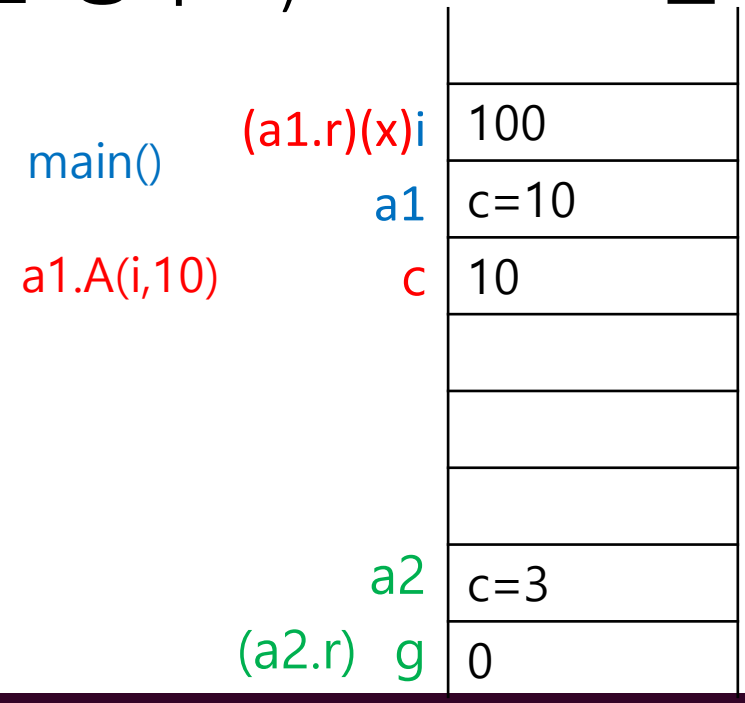
int &x = g;
int &r = x;

```
(gdb) b 9
Breakpoint 1 at 0xd20: file constructor2.cpp,
(gdb) b 14
Breakpoint 2 at 0xda7: file constructor2.cpp,
(gdb) r
Starting program: /home/ejim/C2020/constructo

Breakpoint 1, A::A (this=0x555555756150 <a2>,
    at constructor2.cpp:9
9           cout << this << " : A(int&, int)
(gdb) p &r
$1 = (int *) 0x555555756140 <g>
(gdb) p &g
$2 = (int *) 0x555555756140 <g>
(gdb) c
Continuing.
0x555555756150 : A(int&, int) called

Breakpoint 1, A::A (this=0x7fffffffde60, x=@0
9           cout << this << " : A(int&, int)
(gdb) bt
#0  A::A (this=0x7fffffffde60, x=@0x7fffffffc
#1  0x0000555555554ae1 in main () at construc
(gdb) p &r
$3 = (int *) 0x7fffffffde5c
(gdb) up
#1  0x0000555555554ae1 in main () at construc
20        A a1(i, 10);
(gdb) p &i
$4 = (int *) 0x7fffffffde5c
(gdb) down
#0  A::A (this=0x7fffffffde60, x=@0x7fffffffc
9           cout << this << " : A(int&, int)
(gdb) p &x
$5 = (int *) 0x7fffffffde5c
```

초기화 리스트를 사용해야 하는 경우 2) reference 멤버

```
4  class A{
5    const int c;
6    int &r;
7  public:
8    A(int &x, int c = 0) : c(c), r(x){
9        cout << this << " : A(int&, int) called\n";
10   }
11   ~A(){ cout << this << " : ~A() called\n";}
12   int getR(){     return r;}
13   int getC(){     return c;}
14   void setR(int v){ r = v; }
15 };
16 int g;
17 A a2(g, 3);
18 int main(){
19   int i = 100;
20   A a1(i, 10);
21   cout << a1.getR() << endl; a1.setR(-1);
22   cout << a1.getR() << " i : " << i << endl;
23   cout << a1.getC() << endl;
24   cout << a2.getR() << endl; a2.setR(-2);
25   cout << a2.getR() << " g : " << g << endl;
26   cout << a2.getC() << endl;
27   return 0;
```

int &x = i;
int &r = x;




```
0x55a9be7af150 : A(int&, int) called
0x7fffc12f8c80 : A(int&, int) called
100
-1 i : -1
10
0
-2 g : -2
3
0x7fffc12f8c80 : ~A() called
0x55a9be7af150 : ~A() called
```

```
4  class A{
5    const int c;
6    int &r;
7  public:
8    A(int &x, int c = 0) : c(c), r(x){
9        cout << this << " : A(int&, int) called\n";
10   }
11   ~A(){ cout << this << " : ~A() called\n";}
12   int getR(){     return r;}
13   int getC(){     return c;}
14   void setR(int v){ r = v; }
15 };
16 int g;
17 A a2(g, 3);
18 int main(){
19   int i = 100;
20   A a1(i, 10);
21   cout << a1.getR() << endl; a1.setR(-1);
22   cout << a1.getR() << " i : " << i << endl;
23   cout << a1.getC() << endl;
24   cout << a2.getR() << endl; a2.setR(-2);
25   cout << a2.getR() << " g : " << g << endl;
26   cout << a2.getC() << endl;
27   return 0;
```

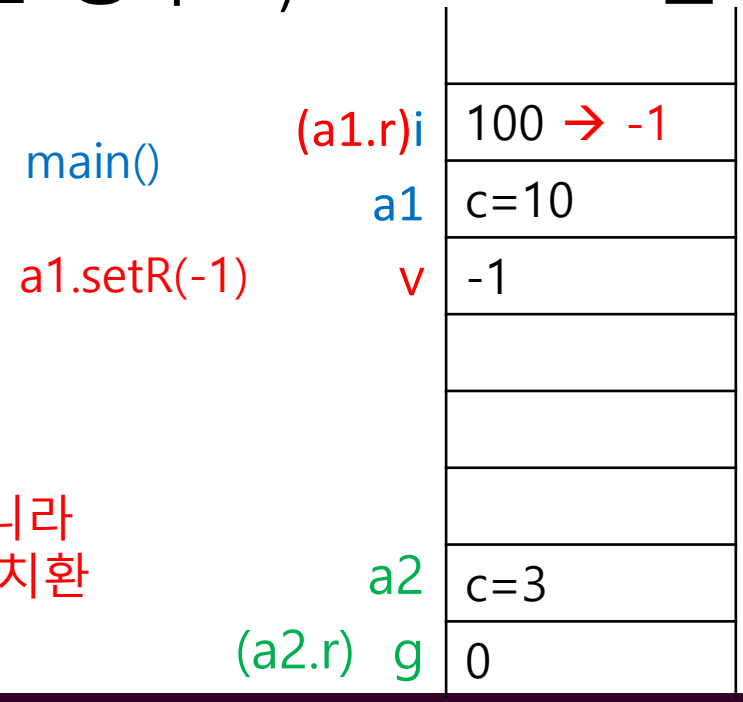
int &x = i;
int &r = x;

```
(gdb) b 9
Breakpoint 1 at 0xd20: file constructor2.cpp,
(gdb) b 14
Breakpoint 2 at 0xda7: file constructor2.cpp,
(gdb) r
Starting program: /home/ejim/C2020/constructo

Breakpoint 1, A::A (this=0x555555756150 <a2>,
    at constructor2.cpp:9
9           cout << this << " : A(int&, int)
(gdb) p &r
$1 = (int *) 0x555555756140 <g>
(gdb) p &g
$2 = (int *) 0x555555756140 <g>
(gdb) c
Continuing.
0x555555756150 : A(int&, int) called

Breakpoint 1, A::A (this=0x7fffffffde60, x=@0
9           cout << this << " : A(int&, int)
(gdb) bt
#0  A::A (this=0x7fffffffde60, x=@0x7fffffffc
#1  0x0000555555554ae1 in main () at construc
(gdb) p &r
$3 = (int *) 0x7fffffffde5c
(gdb) up
#1  0x0000555555554ae1 in main () at construc
20        A a1(i, 10);
(gdb) p &i
$4 = (int *) 0x7fffffffde5c
(gdb) down
#0  A::A (this=0x7fffffffde60, x=@0x7fffffffc
9           cout << this << " : A(int&, int)
(gdb) p &x
$5 = (int *) 0x7fffffffde5c
```

초기화 리스트를 사용해야 하는 경우 2) reference 멤버

```
4  class A{
5    const int c;
6    int &r;
7  public:
8    A(int &x, int c = 0) : c(c), r(x){
9        cout << this << " : A(int&, int) called\n";
10   }
11   ~A(){ cout << this << " : ~A() called\n";}
12   int getR(){     return r;}
13   int getC(){     return c;}
14   void setR(int v){ r = v; }
15 };
16 int g;
17 A a2(g, 3);
18 int main(){
19   int i = 100;
20   A a1(i, 10);
21   cout << a1.getR() << endl; a1.setR(-1);
22   cout << a1.getR() << " i : " << i << endl;
23   cout << a1.getC() << endl;
24   cout << a2.getR() << endl; a2.setR(-2);
25   cout << a2.getR() << " g : " << g << endl;
26   cout << a2.getC() << endl;
27   return 0;
```

reference r 을 초기화하는 것이 아니라
r 이라는 별명을 가진 변수에 값을 치환

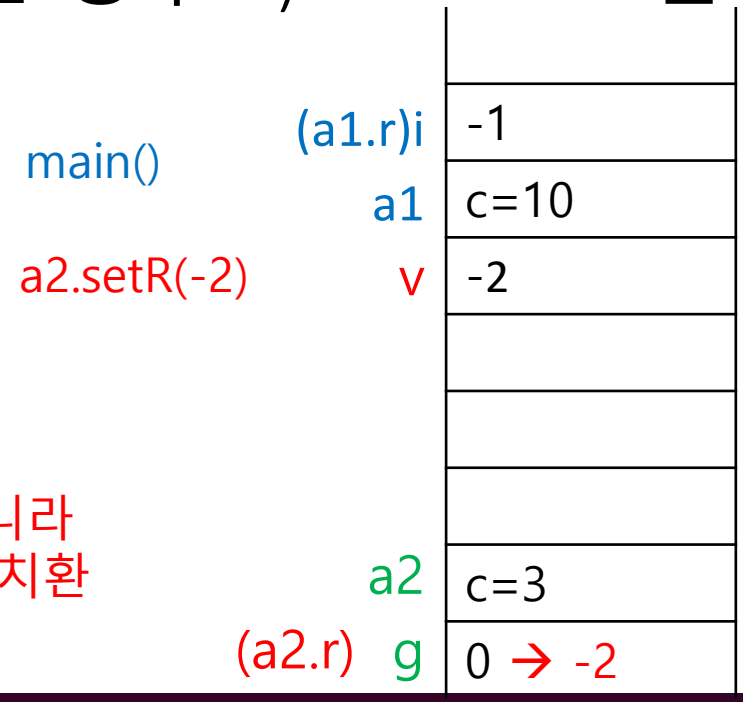



```
0x55a9be7af150 : A(int&, int) called
0x7fffc12f8c80 : A(int&, int) called
100
-1 i : -1
10
0
-2 g : -2
3
0x7fffc12f8c80 : ~A() called
0x55a9be7af150 : ~A() called
```

초기화 리스트를 사용해야 하는 경우 2) reference 멤버

```
4  class A{
5    const int c;
6    int &r;
7  public:
8    A(int &x, int c = 0) : c(c), r(x){
9        cout << this << " : A(int&, int) called\n";
10   }
11   ~A(){ cout << this << " : ~A() called\n";}
12   int getR(){     return r;}
13   int getC(){     return c;}
14   void setR(int v){ r = v; }
15 };
16 int g;
17 A a2(g, 3);
18 int main(){
19   int i = 100;
20   A a1(i, 10);
21   cout << a1.getR() << endl; a1.setR(-1);
22   cout << a1.getR() << " i : " << i << endl;
23   cout << a1.getC() << endl;
24   cout << a2.getR() << endl; a2.setR(-2);
25   cout << a2.getR() << " g : " << g << endl;
26   cout << a2.getC() << endl;
27   return 0;
```

reference r 을 초기화하는 것이 아니라
r 이라는 별명을 가진 변수에 값을 치환

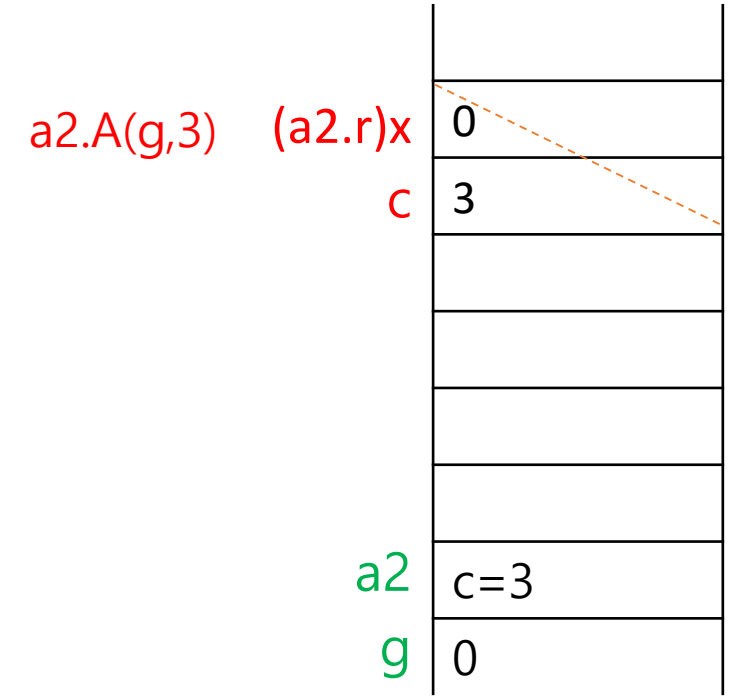



```
0x55a9be7af150 : A(int&, int) called
0x7fffc12f8c80 : A(int&, int) called
100
-1 i : -1
10
0
-2 g : -2
3
0x7fffc12f8c80 : ~A() called
0x55a9be7af150 : ~A() called
```

(2) 레퍼런스 멤버의 초기화 (잘못된 예)

```
4  class A{
5    const int c;
6    int &r;
7  public:  &가 아님
8    A(int x, int c = 0) : c(c), r(x){
9      cout << this << " : A(int, int) called\n";
10   }
11   ~A(){ cout << this << " : ~A() called\n";}
12   int getR(){     return r;}
13   int getC(){     return c;}
14   void setR(int v){ r = v; }
15 };
16 int g;
17 A a2(g, 3);
18 int main(){
19   int i = 100;
20   A a1(i, 10);
21   cout << a1.getR() << endl; a1.setR(-1);
22   cout << a1.getR() << " i : " << i << endl;
23   cout << a1.getC() << endl;
24   cout << a2.getR() << endl; a2.setR(-2);
25   cout << a2.getR() << " g : " << g << endl;
26   cout << a2.getC() << endl;
27   return 0;
```

int x = g;
int &r = x;
x 는 지역 변수




```
0x55d22c086150 : A(int, int) called
0x7fff9aed94a0 : A(int, int) called
100
-1 i : 100
10
32565
-2 g : 0
3
0x7fff9aed94a0 : ~A() called
0x55d22c086150 : ~A() called
```

```
4  class A{
5    const int c;
6    int &r;
7  public:
8    A(int x, int c = 0) : c(c), r(x){
9        cout << this << " : A(int, int) called\n";
10   }
11   ~A(){ cout << this << " : ~A() called\n";}
12   int getR(){     return r;}
13   int getC(){     return c;}
14   void setR(int v){ r = v; }
15 };
16 int g;
17 A a2(g, 3);
18 int main(){
19   int i = 100;
20   A a1(i, 10);
21   cout << a1.getR() << endl; a1.setR(-1);
22   cout << a1.getR() << " i : " << i << endl;
23   cout << a1.getC() << endl;
24   cout << a2.getR() << endl; a2.setR(-2);
25   cout << a2.getR() << " g : " << g << endl;
26   cout << a2.getC() << endl;
27   return 0;
```

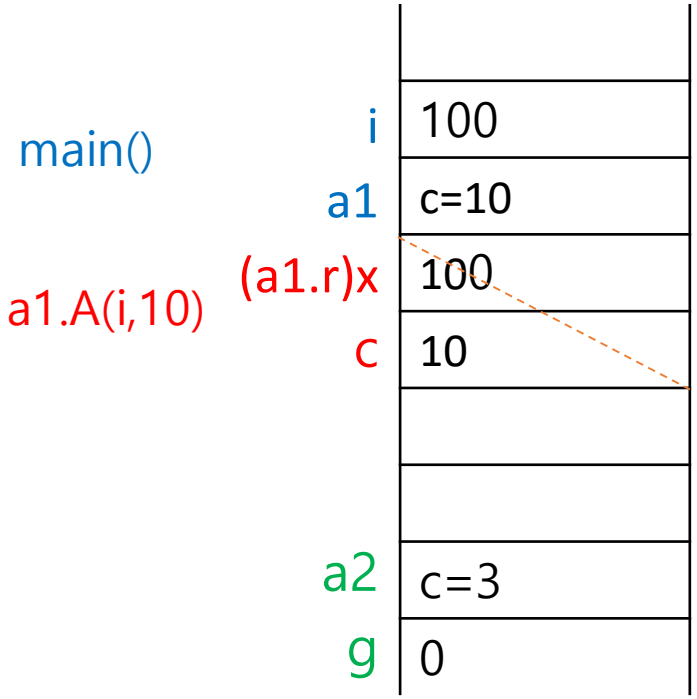
int x = g;
int &r = x;
x 는 지역 변수

```
(gdb) b 9
Breakpoint 1 at 0xd1d: file constructor2.cpp,
(gdb) b 14
Breakpoint 2 at 0xda3: file constructor2.cpp,
(gdb) r
Starting program: /home/ejim/C2020/constructo

Breakpoint 1, A::A (this=0x555555756150 <a2>,
9             cout << this << " : A(int, int)
(gdb) p &r
$1 = (int *) 0x7fffffffde14
(gdb) p &g
$2 = (int *) 0x555555756140 <g>
(gdb) c
Continuing.
0x555555756150 : A(int, int) called

Breakpoint 1, A::A (this=0x7fffffffde60, x=10
9             cout << this << " : A(int, int)
(gdb) bt
#0  A::A (this=0x7fffffffde60, x=100, c=10) a
#1  0x0000555555554adf in main () at construc
(gdb) p &r
$3 = (int *) 0x7fffffffde34
(gdb) up
#1  0x0000555555554adf in main () at construc
20        A a1(i, 10);
(gdb) p &i
$4 = (int *) 0x7fffffffde5c
(gdb) down
#0  A::A (this=0x7fffffffde60, x=100, c=10) a
9             cout << this << " : A(int, int)
(gdb) p &x
$5 = (int *) 0x7fffffffde34
```

(2) 레퍼런스 멤버의 초기화 (잘못된 예)

```
4  class A{
5    const int c;
6    int &r;
7  public:  &가 아님
8    A(int x, int c = 0) : c(c), r(x){
9      cout << this << " : A(int, int) called\n";
10   }
11   ~A(){ cout << this << " : ~A() called\n";}
12   int getR(){     return r;}
13   int getC(){     return c;}
14   void setR(int v){ r = v; }
15 };
16 int g;
17 A a2(g, 3);
18 int main(){
19   int i = 100;
20   A a1(i, 10);
21   cout << a1.getR() << endl; a1.setR(-1);
22   cout << a1.getR() << " i : " << i << endl;
23   cout << a1.getC() << endl;
24   cout << a2.getR() << endl; a2.setR(-2);
25   cout << a2.getR() << " g : " << g << endl;
26   cout << a2.getC() << endl;
27   return 0;
```

int x = i;
int &r = x;
x 는 지역 변수



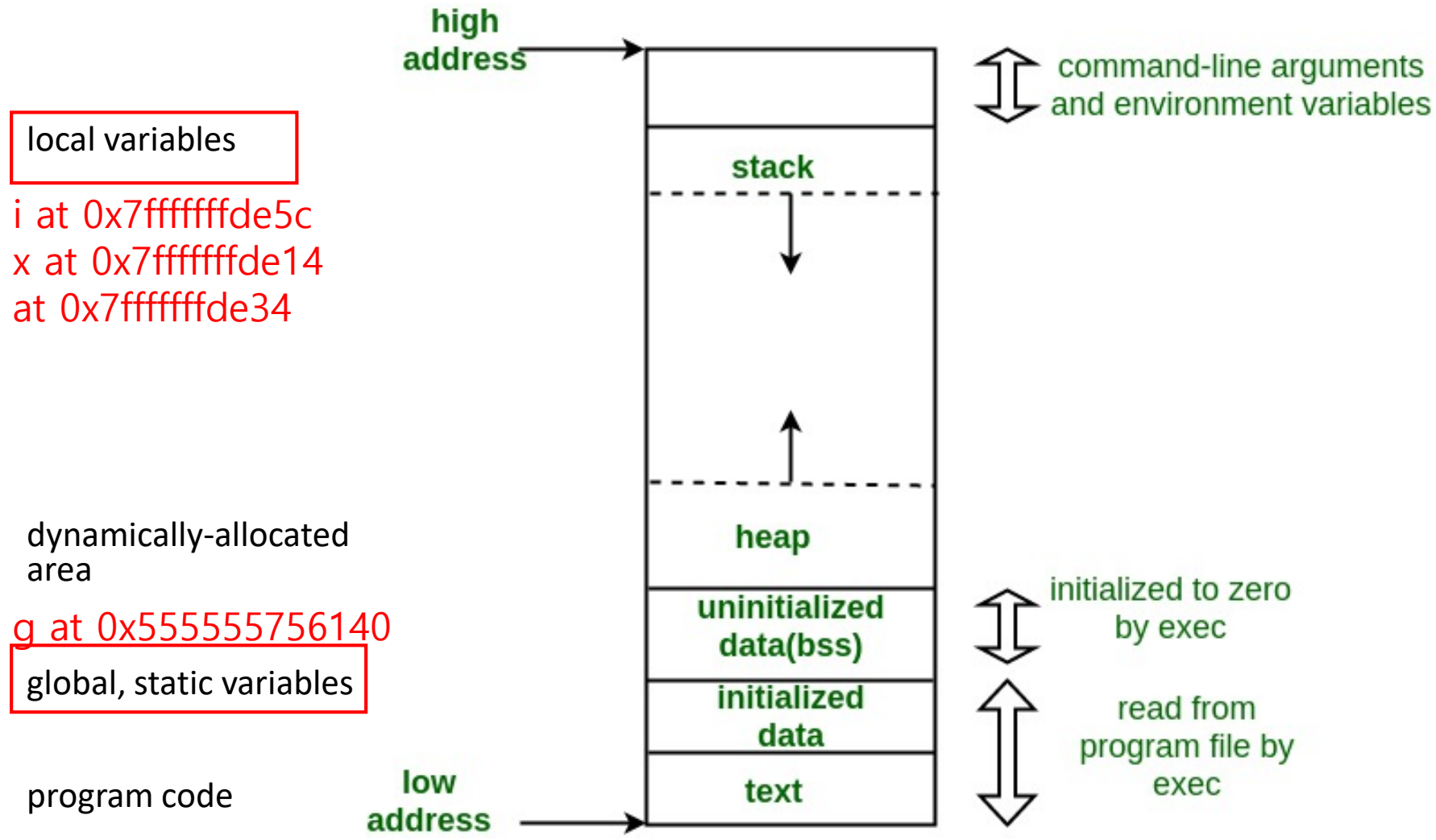

```
0x55d22c086150 : A(int, int) called
0x7fff9aed94a0 : A(int, int) called
100
-1 i : 100
10
32565
-2 g : 0
3
0x7fff9aed94a0 : ~A() called
0x55d22c086150 : ~A() called
```

```
4  class A{
5    const int c;
6    int &r;
7  public:
8    A(int x, int c = 0) : c(c), r(x){
9        cout << this << " : A(int, int) called\n";
10   }
11   ~A(){ cout << this << " : ~A() called\n";}
12   int getR(){     return r;}
13   int getC(){     return c;}
14   void setR(int v){ r = v; }
15 };
16 int g;
17 A a2(g, 3);
18 int main(){
19   int i = 100;
20   A a1(i, 10);
21   cout << a1.getR() << endl; a1.setR(-1);
22   cout << a1.getR() << " i : " << i << endl;
23   cout << a1.getC() << endl;
24   cout << a2.getR() << endl; a2.setR(-2);
25   cout << a2.getR() << " g : " << g << endl;
26   cout << a2.getC() << endl;
27   return 0;
```

int x = i;
int &r = x;
x 는 지역 변수

```
(gdb) b 9
Breakpoint 1 at 0xd1d: file constructor2.cpp,
(gdb) b 14
Breakpoint 2 at 0xda3: file constructor2.cpp,
(gdb) r
Starting program: /home/ejim/C2020/constructo

Breakpoint 1, A::A (this=0x555555756150 <a2>,
9             cout << this << " : A(int, int)
(gdb) p &r
$1 = (int *) 0x7fffffffde14
(gdb) p &g
$2 = (int *) 0x555555756140 <g>
(gdb) c
Continuing.
0x555555756150 : A(int, int) called

Breakpoint 1, A::A (this=0x7fffffffde60, x=10
9             cout << this << " : A(int, int)
(gdb) bt
#0  A::A (this=0x7fffffffde60, x=100, c=10) a
#1  0x0000555555554adf in main () at construc
(gdb) p &r
$3 = (int *) 0x7fffffffde34
(gdb) up
#1  0x0000555555554adf in main () at construc
20        A a1(i, 10);
(gdb) p &i
$4 = (int *) 0x7fffffffde5c
(gdb) down
#0  A::A (this=0x7fffffffde60, x=100, c=10) a
9             cout << this << " : A(int, int)
(gdb) p &x
$5 = (int *) 0x7fffffffde34
```

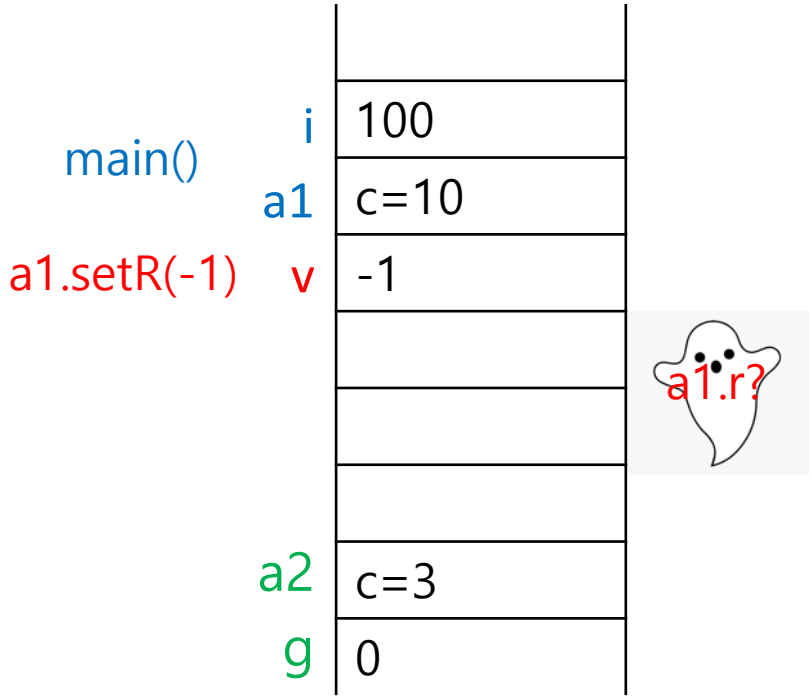
Typical Memory Layout of a C program



(2) 레퍼런스 멤버의 초기화 (잘못된 예)

```
4  class A{
5    const int c;
6    int &r;
7  public:  &가 아님
8    A(int x, int c = 0) : c(c), r(x){
9      cout << this << " : A(int, int) called\n";
10   }
11   ~A(){ cout << this << " : ~A() called\n";}
12   int getR(){     return r;}
13   int getC(){     return c;}
14   void setR(int v){ r = v; }
15 };
16 int g;
17 A a2(g, 3);
18 int main(){
19   int i = 100;
20   A a1(i, 10);
21   cout << a1.getR() << endl; a1.setR(-1);
22   cout << a1.getR() << " i : " << i << endl;
23   cout << a1.getC() << endl;
24   cout << a2.getR() << endl; a2.setR(-2);
25   cout << a2.getR() << " g : " << g << endl;
26   cout << a2.getC() << endl;
27   return 0;
```

reference r 을 초기화하는 것이 아니라
r 이라는 별명을 가진 변수에 값을 치환

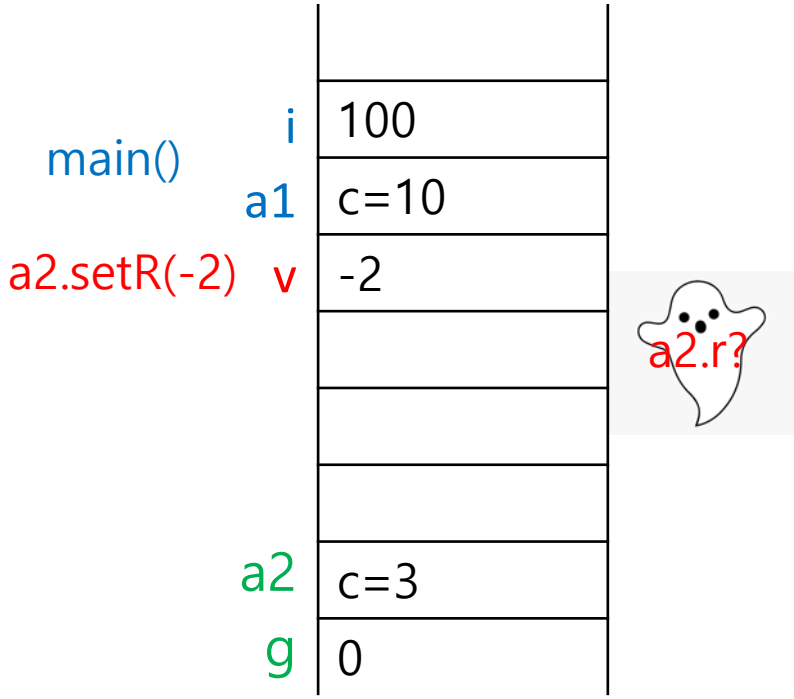



```
0x55d22c086150 : A(int, int) called
0x7fff9aed94a0 : A(int, int) called
100
-1 i : 100
10
32565
-2 g : 0
3
0x7fff9aed94a0 : ~A() called
0x55d22c086150 : ~A() called
```

(2) 레퍼런스 멤버의 초기화 (잘못된 예)

```
4  class A{
5    const int c;
6    int &r;
7  public:  &가 아님
8    A(int x, int c = 0) : c(c), r(x){
9      cout << this << " : A(int, int) called\n";
10   }
11   ~A(){ cout << this << " : ~A() called\n";}
12   int getR(){     return r;}
13   int getC(){     return c;}
14   void setR(int v){ r = v; }
15 };
16 int g;
17 A a2(g, 3);
18 int main(){
19   int i = 100;
20   A a1(i, 10);
21   cout << a1.getR() << endl; a1.setR(-1);
22   cout << a1.getR() << " i : " << i << endl;
23   cout << a1.getC() << endl;
24   cout << a2.getR() << endl; a2.setR(-2);
25   cout << a2.getR() << " g : " << g << endl;
26   cout << a2.getC() << endl;
27   return 0;
```

reference r 을 초기화하는 것이 아니라
r 이라는 별명을 가진 변수에 값을 치환




```
0x55d22c086150 : A(int, int) called
0x7fff9aed94a0 : A(int, int) called
100
-1 i : 100
10
32565
-2 g : 0
3
0x7fff9aed94a0 : ~A() called
0x55d22c086150 : ~A() called
```

초기화 리스트를 사용하는 경우 3) 객체 멤버

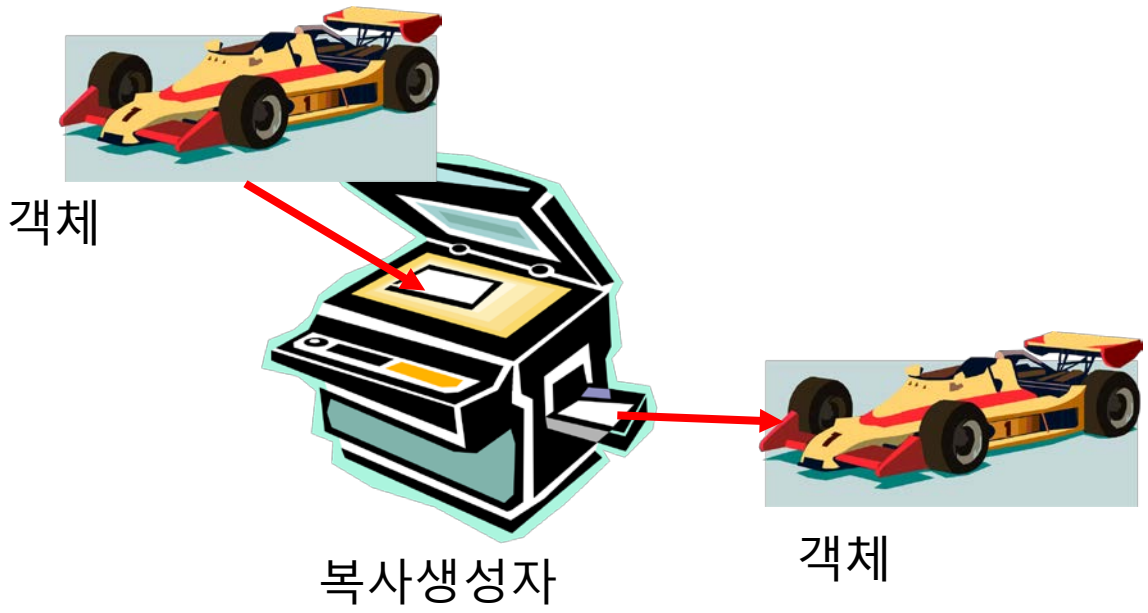
```
4  class Point{
5    int x,y;
6  public:
7    Point(int x,int y):x(x){
8       this->y = y;
9    }
10 };
11 class Circle{
12   Point center;
13   int radius;
14 public:
15   Circle(int x,int y, int r):center(x,y), radius(r){}
16 };
17 int main(){
18   Circle c1(4,5,2);
19   return 0;
20 }
```

Point 생성자 호출

```
Reading symbols from constructor4...done.
(gdb) b main
Breakpoint 1 at 0x7b2: file constructor4.cpp, line 17.
(gdb) r
Starting program: /home/ejim/C2020/constructor4

Breakpoint 1, main () at constructor4.cpp:17
17      int main(){
(gdb) n
18          Circle c1(4,5,2);
(gdb) p c1
$1 = {center = {x = 21845, y = -8336}, radius = 32767}
(gdb) n
19          return 0;
(gdb) p c1
$2 = {center = {x = 4, y = 5}, radius = 2}
```

복사 생성자

- 한 객체의 내용을 다른 객체로 복사하여서 생성



복사 생성자의 특징

- 자신과 같은 타입의 객체를 매개 변수로 받는 생성자.
- 복사 생성자가 없으면 자동으로 모든 멤버 변수의 값을 복사하는 디폴트 복사 생성자가 생성된다.

```
Car(Car& obj);

Car(const Car& obj);
```

Default Copy Constructor

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

A2 에 대해서는 default copy constructor 호출

```
0x7ffe48c9ce90 : A(int) called
10
10
0x7ffe48c9ce94 : ~A() called
0x7ffe48c9ce90 : ~A() called
```

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

copy constructor

```
0x7ffc6f85d860 : A(int) called
0x7ffc6f85d864 : A(const A&) called
10
10
0x7ffc6f85d864 : ~A() called
0x7ffc6f85d860 : ~A() called
```

복사 생성자가 호출되는 경우

- 기존의 객체의 내용을 복사하여서 새로운 객체를 만드는 경우
- 객체를 값으로 매개 변수로 전달하는 경우
- (객체를 값으로 반환하는 경우) ← compiler 가 복사 생성자를 호출하지 않도록 최적화하는 경우도 있음

복사 생성자

```
4  class A{
5    int p;
6  public:
7    A(const A& obj){ copy constructor
8       p = obj.p;
9       cout << this << " : A(const A& " << &obj <<")\n";
10   }
11   A(int v = 0) : p(v){
12      cout << this << " : A(int " << v <<")\n";
13   }
14   ~A(){
15     cout << this << " : ~A()\n";
16   }
17   int getP(){
18      return p;
19   }
20   void setP(int v){
21      p = (v<0)? 0 : v;
22   }
23   A return_this(){
24      return *this;
25   }
26 };
```

```
27 A plusOne(A c){
28   A result(c.getP()+1);
29   return result;
30 }
31 int main(){
32   A a1(10), a2(a1);
33   cout << a1.getP() << endl;
34   cout << a2.getP() << endl;
35   a2 = plusOne(a1).return_this();
36   cout << a2.getP() << endl;
37   return 0;
```

A c(a1)

a1
a2
c
result
plusOne(a1)
~plusOne(a1)
~result
~c
~a2
~a1

```
0x7ffc6f35f4f4 : A(int 10)
0x7ffc6f35f4f8 : A(const A& 0x7ffc6f35f4f4)
10
10
0x7ffc6f35f4fc : A(const A& 0x7ffc6f35f4f4)
0x7ffc6f35f500 : A(int 11)
0x7ffc6f35f504 : A(const A& 0x7ffc6f35f500)
0x7ffc6f35f504 : ~A()
0x7ffc6f35f500 : ~A()
0x7ffc6f35f4fc : ~A()
11
0x7ffc6f35f4f8 : ~A()
0x7ffc6f35f4f4 : ~A()
```

Microsoft VisualStudio C++

```
4  class A{
5    int p;
6  public:
7    A(const A& obj){ copy constructor
8       p = obj.p;
9       cout << this << " : A(const A& " << &obj <<")\n";
10   }
11   A(int v = 0) : p(v){
12      cout << this << " : A(int " << v <<")\n";
13   }
14   ~A(){
15     cout << this << " : ~A()\n";
16   }
17   int getP(){
18      return p;
19   }
20   void setP(int v){
21      p = (v<0)? 0 : v;
22   }
23   A return_this(){
24      return *this;
25   }
26 };
```

```
27 A plusOne(A c){ A c(a1)
28   A result(c.getP()+1);
29   return result;
30 }
31 int main(){
32   A a1(10), a2(a1);
33   cout << a1.getP() << endl;
34   cout << a2.getP() << endl;
35   a2 = plusOne(a1).return_this();
36   cout << a2.getP() << endl;
37   return 0;
```

a1
a2

c
result
plusOne(a1)
~result
~c
A return_this()
~A return_this()
~plusOne(a1)

~a2
~a1

Microsoft Visual Studio 디버그...

```
004FFB78 : A(int 10)
004FFB6C : A(const A& 004FFB78)
10
10
004FFA50 : A(const A& 004FFB78)
004FFA2C : A(int 11)
004FFA94 : A(const A& 004FFA2C)
004FFA2C : ~A()
004FFA50 : ~A()
004FFA88 : A(const A& 004FFA94)
004FFA88 : ~A()
004FFA94 : ~A()
11
004FFB6C : ~A()
004FFB78 : ~A()
```

다음 예제인

Student class 에서는 c string functions strncpy(), strlen() 을 사용한다.

header

<cstring> (string.h)

C Strings

This header file defines several functions to manipulate *C strings* and arrays.

Functions

Copying:

memcpy	Copy block of memory (function)
memmove	Move block of memory (function)
strcpy	Copy string (function)
strncpy	Copy characters from string (function)

Concatenation:

strcat	Concatenate strings (function)
strncat	Append characters from string (function)

Comparison:

memcmp	Compare two blocks of memory (function)
strcmp	Compare two strings (function)
strcoll	Compare two strings using locale (function)
strncmp	Compare characters of two strings (function)
strxfrm	Transform string using locale (function)

function

strcpy

<cstring>

```
char * strcpy ( char * destination, const char * source );
```

Copy string

Copies the C string pointed by *source* into the array pointed by *destination*, including the terminating null character (and stopping at that point).

To avoid overflows, the size of the array pointed by *destination* shall be long enough to contain the same C string as *source* (including the terminating null character), and should not overlap in memory with *source*.

Parameters

`destination`
: Pointer to the destination array where the content is to be copied.

`source`
: C string to be copied.

Return Value

destination is returned.

Example

```
1  /* strcpy example */
2  #include <stdio.h>
3  #include <string.h>
4
5  int main ()
6  {
7    char str1[]="Sample string";
8    char str2[40];
9    char str3[40];
10   strcpy (str2,str1);
11   strcpy (str3,"copy successful");
12   printf ("str1: %s\nstr2: %s\nstr3: %s\n",str1,str2,str3);
13   return 0;
14 }
```

Edit & Run

function

strncpy

<cstring>

```
char * strncpy ( char * destination, const char * source, size_t num );
```

Copy characters from string

Copies the first *num* characters of *source* to *destination*. If the end of the *source* C string (which is signaled by a null-character) is found before *num* characters have been copied, *destination* is padded with zeros until a total of *num* characters have been written to it.

No null-character is implicitly appended at the end of *destination* if *source* is longer than *num*. Thus, in this case, *destination* shall not be considered a null terminated C string (reading it as such would overflow).

destination and *source* shall not overlap (see memmove for a safer alternative when overlapping).

Parameters

`destination`
: Pointer to the destination array where the content is to be copied.

`source`
: C string to be copied.

`num`
: Maximum number of characters to be copied from *source*.
size_t is an unsigned integral type.

Return Value

destination is returned.

function

strlen

<cstring>

```
size_t strlen ( const char * str );
```

Get string length

Returns the length of the C string *str*.

The length of a C string is determined by the terminating null-character: A *C string* is as long as the number of characters between the beginning of the string and the terminating null character (without including the terminating null character itself).

This should not be confused with the size of the array that holds the string. For example:

```
char mystr[100]="test string";
```

defines an array of characters with a size of 100 `chars`, but the C string with which *mystr* has been initialized has a length of only 11 characters. Therefore, while `sizeof(mystr)` evaluates to `100`, `strlen(mystr)` returns `11`.

In C++, char_traits::length implements the same behavior.

Parameters

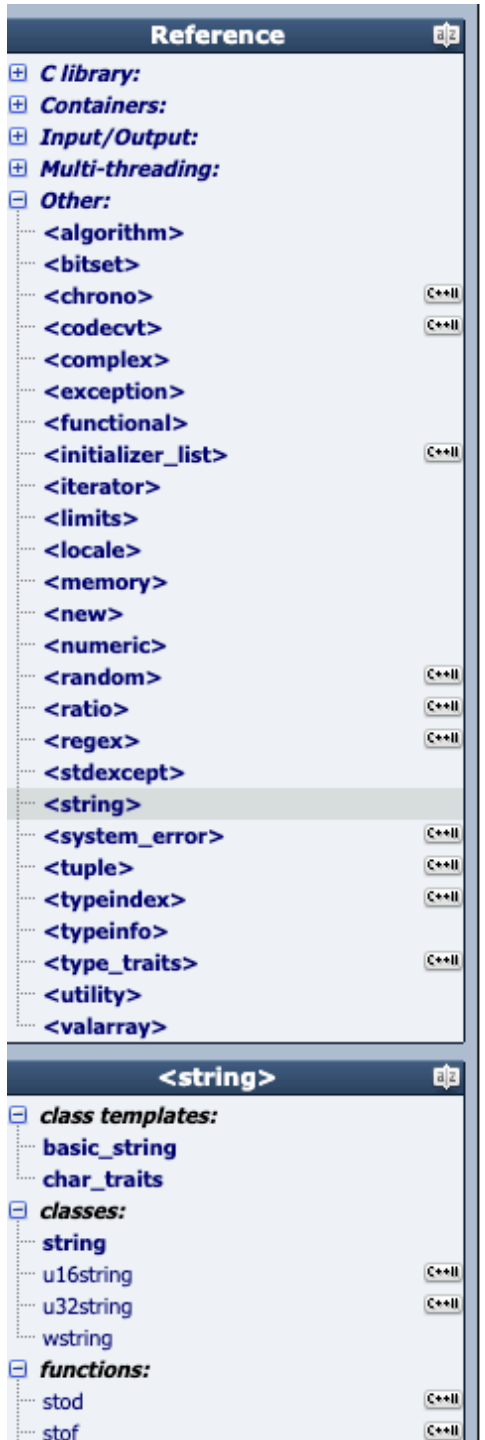
`str`

C string.

<string> C++ string class 선언

vs.

<cstring> : C 문자열 함수들



header

<string>

Strings

This header introduces string types, character traits and a set of converting functions:

Class templates

basic_string	Generic string class (class template)
char_traits	Character traits (class template)

Class instantiations

string	String class (class)
u16string C++11	String of 16-bit characters (class)
u32string C++11	String of 32-bit characters (class)
wstring	Wide string (class)

fx Functions

Convert from strings

stoi C++11	Convert string to integer (function template)
stol C++11	Convert string to long int (function template)
stoul C++11	Convert string to unsigned integer (function template)
stoll C++11	Convert string to long long (function template)
stoull C++11	Convert string to unsigned long long (function template)
stof C++11	Convert string to float (function template)
stod C++11	Convert string to double (function template)
stold C++11	Convert string to long double (function template)

Convert to strings

to_string C++11	Convert numerical value to string (function)
to_wstring C++11	Convert numerical value to wide string (function)

Range access

begin C++11	Iterator to beginning (function template)
end C++11	Iterator to end (function template)

얕은 복사 문제

- 멤버가 포인터 형인 경우 멤버 변수의 값만 복사하면 안되는 경우가 발생한다.
- 얕은 복사(shallow copy) 문제

```cpp
#include <iostream>
#include <cstring>
using namespace std;
#define NAME_LEN 100

class Student{
  int id;
  char *name;
public:
  Student(int n,char *s){
     id = n;
     name = new char[NAME_LEN];
     strncpy(name, s, NAME_LEN);
     name[NAME_LEN-1] = '\0';
  }
  ~Student(){ delete[] name;}
  Student(const Student& st){
     id = st.id;
     name = st.name;
  }
  void setName(char *s){
     strncpy(name, s, NAME
     name[NAME_LEN-1] = '
  }
  void print(){
     cout << this << " ] "
     cout << " id : " << i
  }
};
```

```cpp
int main(){
  Student s1(20100001,"Kim"), s2(s1);
  s1.print(); s2.print();
  s1.setName("Park");
  s1.print(); s2.print();
  return 0;
}
```

문자열 상수

const 인자로 선언하여 warning error 를 없앤다.

```
g++ -g -o shallow1 shallow1.cpp
shallow1.cpp: In function ‘int main()’:
shallow1.cpp:30:28: warning: ISO C++ forbids converting a string constant to ‘char*’ [-Wwrite-strings]
   Student s1(20100001,"Kim"), s2(s1);
                            ^
shallow1.cpp:32:20: warning: ISO C++ forbids converting a string constant to ‘char*’ [-Wwrite-strings]
   s1.setName("Park");
                    ^
```

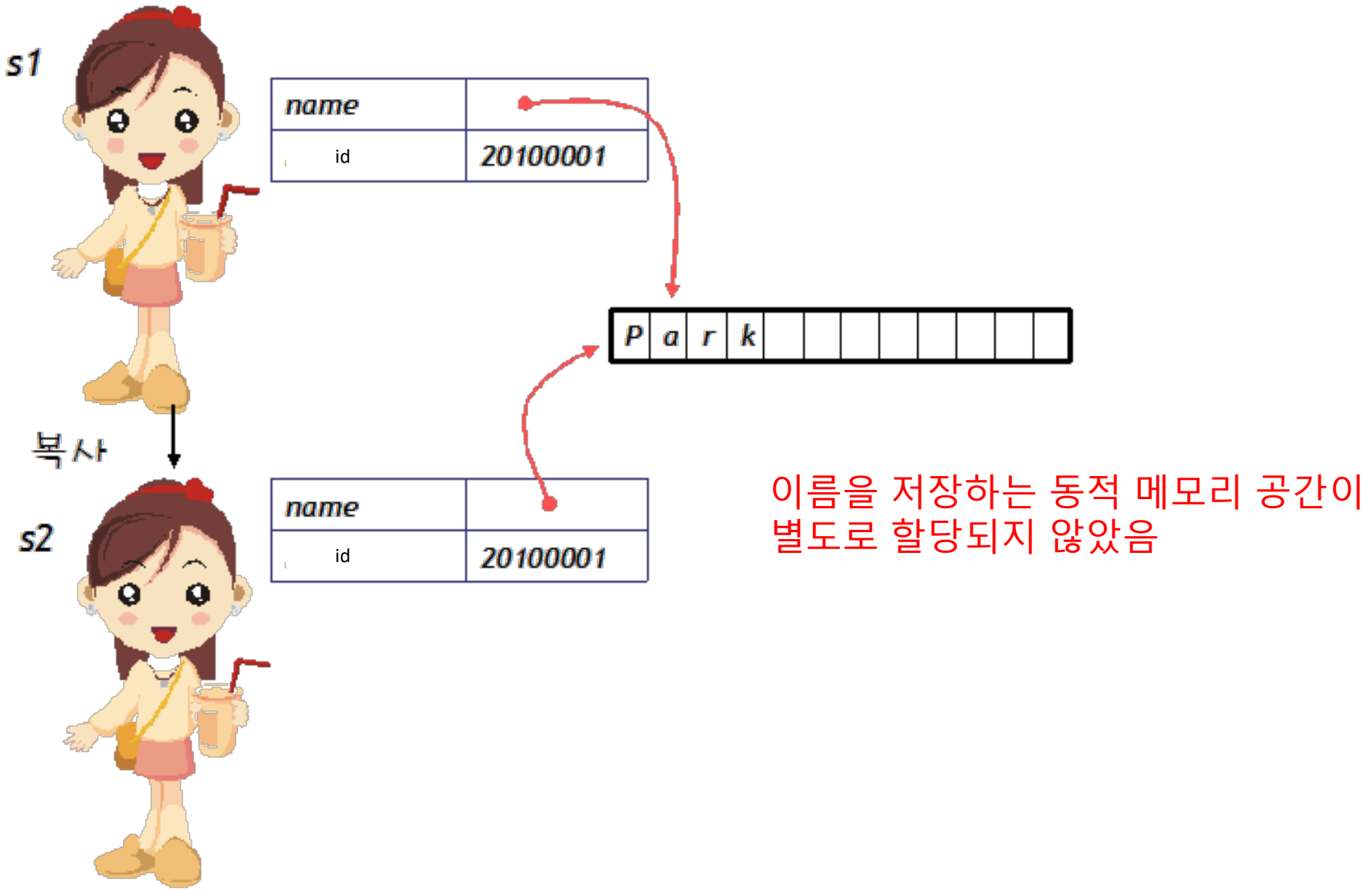
```
6  class Student{
7    int id;
8    char *name;
9  public:
10   Student(int n,const char *s){
11      id = n;
12      name = new char[NAME_LEN];
13      strncpy(name, s, NAME_LEN);
14      name[NAME_LEN-1] = '\0';
15   }
16   ~Student(){ delete[] name;}
17   Student(const Student& st){
18      id = st.id;
19      name = st.name;
20   }
21   void setName(const char *s){
22      strncpy(name, s, NAME_LEN);
23      name[NAME_LEN-1] = '\0';
24   }
25   void print(){
26      cout << this << " ] " << "name : " << name;
27      cout << " id : " << id << endl;
28   }
29 };
```

얕은 복사를 하는 복사 생성자

```
29 int main(){
30   Student s1(20100001,"Kim"), s2(s1);
31   s1.print(); s2.print();
32   s2.setName("Park");
33   s1.print(); s2.print();
34   return 0;
35 }
```

```
0x7ffcfdd49bd0 ] name : Kim id : 20100001
0x7ffcfdd49be0 ] name : Kim id : 20100001
0x7ffcfdd49bd0 ] name : Park id : 20100001
0x7ffcfdd49be0 ] name : Park id : 20100001
```

Shallow Copy 의 문제점

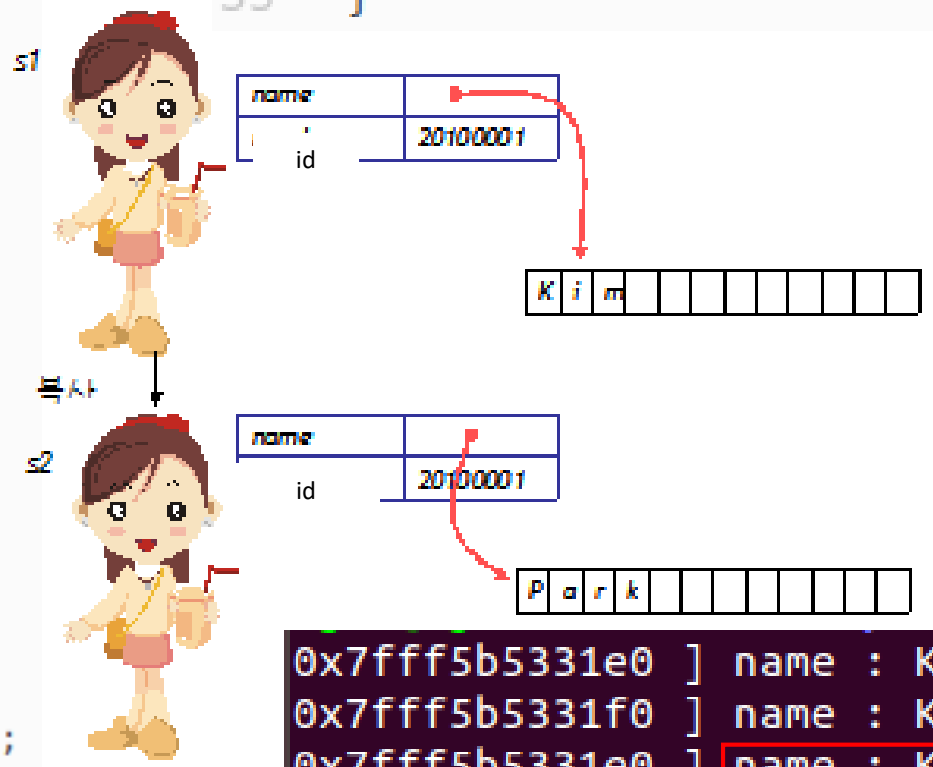


이름을 저장하는 동적 메모리 공간이
별도로 할당되지 않았음

Deep Copy

```
4  #define NAME_LEN 100
5
6  class Student{
7    int id;
8    char *name;
9  public:
10   Student(int n,const char *s){
11      id = n;
12      name = new char[NAME_LEN];
13      strncpy(name, s, NAME_LEN);
14      name[NAME_LEN-1] = '\0';
15   }
16   ~Student(){ delete[] name;}
17   Student(const Student& st){
18      id = st.id;
19      name = new char[NAME_LEN];
20      strncpy(name, st.name, NAME_LEN);
21      name[NAME_LEN-1] = '\0';
22   }
23   void setName(const char *s){
24      strncpy(name, s, NAME_LEN);
25      name[NAME_LEN-1] = '\0';
26   }
27   void print(){
28      cout << this << " ] " << "name : " << name;
29      cout << " id : " << id << endl;
```

```
29  int main(){
30    Student s1(20100001,"Kim"), s2(s1);
31    s1.print(); s2.print();
32    s2.setName("Park");
33    s1.print(); s2.print();
34    return 0;
35  }
```




```
0x7fff5b5331e0 ] name : Kim id : 20100001
0x7fff5b5331f0 ] name : Kim id : 20100001
0x7fff5b5331e0 ] name : Kim id : 20100001
0x7fff5b5331f0 ] name : Park id : 20100001
```

디폴트 멤버 함수

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

자동으로 추가된다.

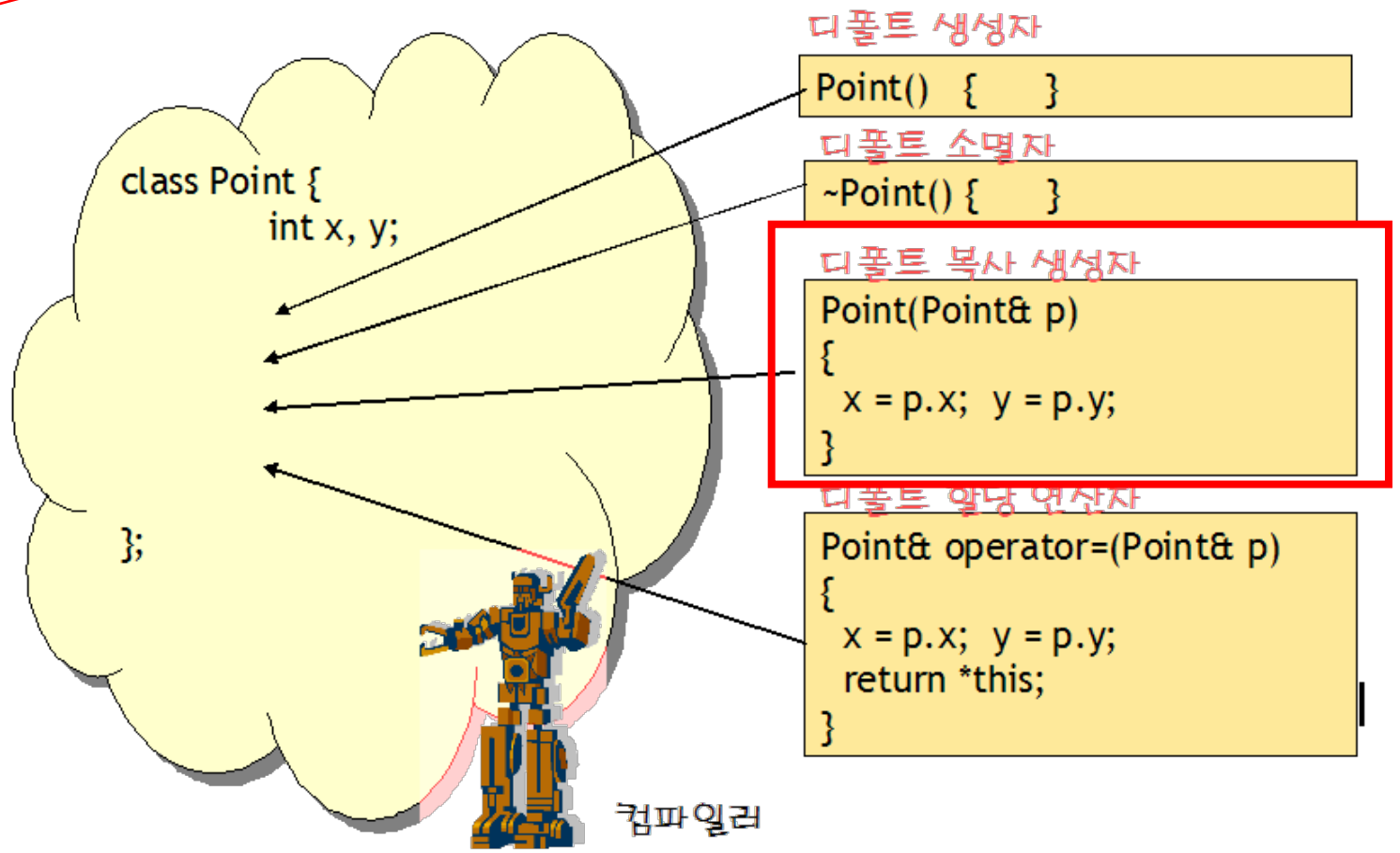
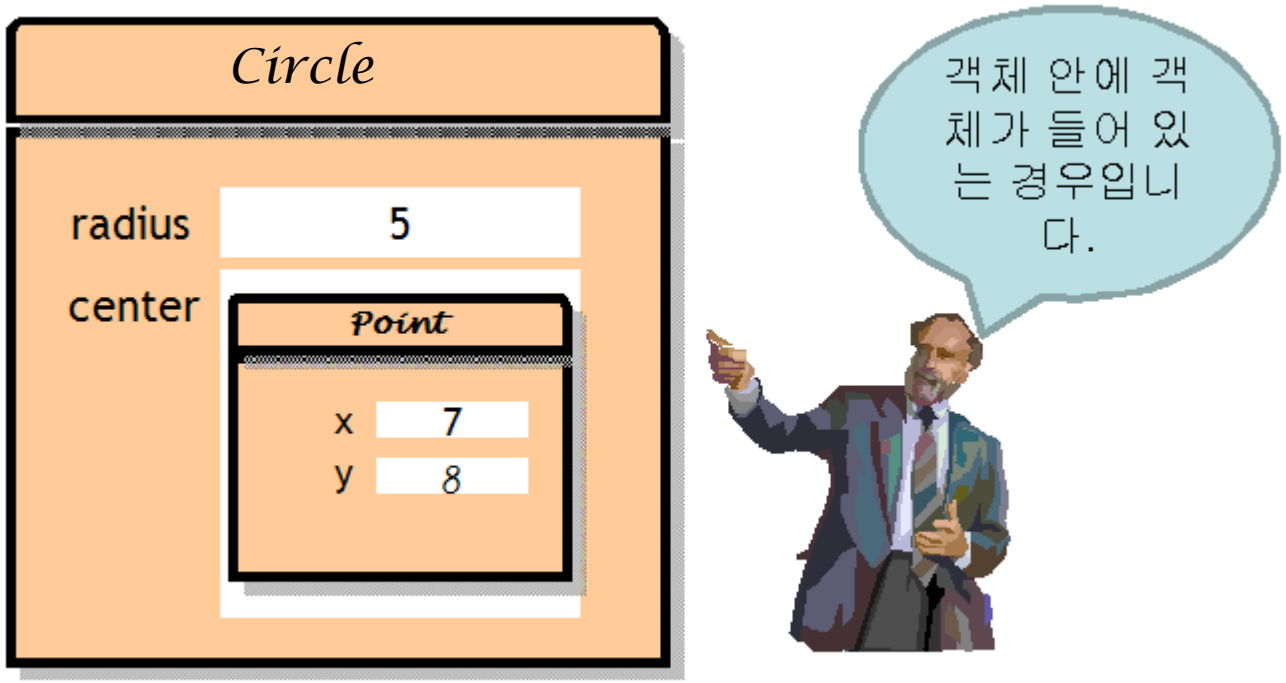


그림 10.7 디폴트 멤버 함수의 추가

예제

- Circle 객체 안에 Point 객체가 들어 있는 경우



```
4  class Point{
5    int x,y;
6  public:
7    Point(int x=0,int y=0):x(x){
8       this->y = y;
9    }
10   void print(){
11      cout << "( " << x << ", " << y << " ) ";
12   }
13 };
14 class Circle{
15   Point center;
16   int radius;
17 public:
18   Circle(int r):center(0,0), radius(r){}
19   Circle(Point p = Point(0,0), int r=0):center(p), radius(r){}
20   Circle(int x,int y, int r=0):center(x,y), radius(r){}
21   void print(){
22      center.print();
23      cout << " : " << radius << endl;
24   }
25 };
26 int main(){
27   Point p(3,4);
28   Circle c1;  c1.print();
29   Circle c2(2);  c2.print();
30   Circle c3(-1,-2);  c3.print();
31   Circle c4(4,5,1);  c4.print();
32   Circle c5(p,5);  c5.print();
```

c2

default constructor : c1, c5

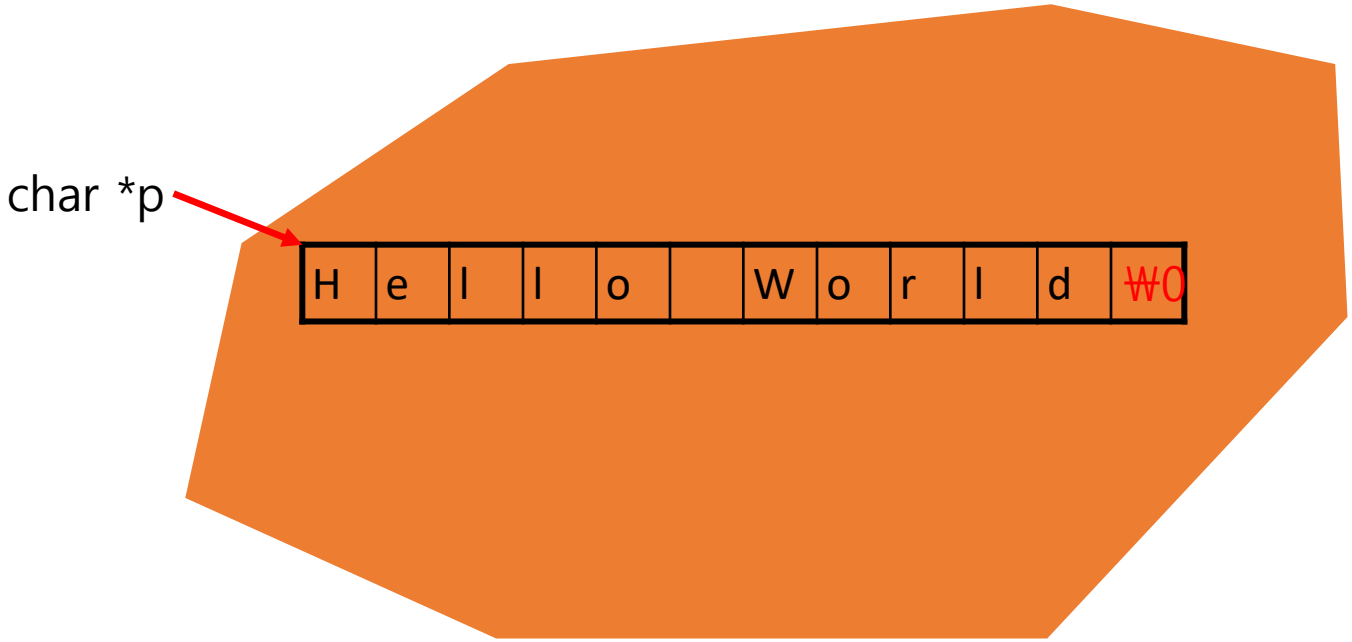
c3, c4

c1
c2
c3
c4
c5

```
( 0, 0 )   : 0
( 0, 0 )   : 2
( -1, -2 )   : 0
( 4, 5 )   : 1
( 3, 4 )   : 5
```

예제

- 문자열을 클래스로 작성해보자.
- C++ string class 와 유사



class

std::string


```
typedef basic_string<char> string;
```

String class

Strings are objects that represent sequences of characters.

The standard `string` class provides support for such objects with an interface similar to that of a standard container of bytes, but adding features specifically designed to operate with strings of single-byte characters.

The `string` class is an instantiation of the basic_string class template that uses `char` (i.e., bytes) as its *character type*, with its default char_traits and allocator types (see basic_string for more info on the template).

Note that this class handles bytes independently of the encoding used: If used to handle sequences of multi-byte or variable-length characters (such as UTF-8), all members of this class (such as length or size), as well as its iterators, will still operate in terms of bytes (not actual encoded characters).

Member types

member type	definition
value_type	char
traits_type	char_traits<char>
allocator_type	allocator<char>
reference	char&
const_reference	const char&
pointer	char*
const_pointer	const char*
iterator	a random access iterator to char (convertible to const_iterator)
const_iterator	a random access iterator to const char
reverse_iterator	reverse_iterator<iterator>
const_reverse_iterator	reverse_iterator<const_iterator>
difference_type	ptrdiff_t
size_type	size_t

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

class MyString{
  char *p;
public:
  MyString(const char *str=NULL);
  MyString(MyString& s);
  ~MyString();
  void print();
  int size();
};
MyString::MyString(const char *str){
   if (!str){
     p = new char[1];
     p[0] = '\0';
     return;
   }
   p = new char[strlen(str)+1];
   strcpy(p, str);
}
MyString::MyString(MyString& s){
  p = new char[s.size()+1]; // deep copy
   strcpy(p, s.p);
}
MyString::~MyString(){ delete[] p; }
void MyString::print(){ cout << p << endl; }
int MyString::size(){ return strlen(p); }
```

default constructor

default constructor

copy constructor

```cpp
int main(){
   MyString s1;
   MyString s2("C++");
   MyString s3(s2);
   s1.print();
   s2.print();
   s3.print();
   return 0;
}
```

```
C++
C++
```

```
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

```
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 }
```

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

```
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
```

const 포인터

- const int *p1;
- p1은 const int에 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  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 : Kvector class 의 멤버 함수들을 class 외부에 구현하라.

(1) 생성자 Kvector(int sz, int value) : sz 개의 정수 배열을 m 에 할당받아 모든 원소의 값을 value 로 초기화한다.
sz 가 0인 경우는 m 은 NULL 이 된다.

(2) 깊은 복사를 구현하는 복사 생성자

main() 함수에 대해서 출력이 다음과 같아야 한다.

```
0x7ffd02e928b0 : Kvector(int)
0 0 0
0x7ffd02e928c0 : Kvector(int)
9 9
0x7ffd02e928d0 : Kvector(Kvector&)
9 9

9 9
0x7ffd02e928d0 : ~Kvector()
0x7ffd02e928c0 : ~Kvector()
0x7ffd02e928b0 : ~Kvector()
```