

# 객체지향 프로그래밍

## 2. C++ 기본

### 1. 기본 요소, 화면 출력

I, 주석문 : 여러 줄 : '/\*' 와 '\*/'

한 줄 : '//'

II, main() : C++ 프로그램의 실행 시작점

· ANSI C++ 표준에서 main() 함수의 리턴 타입은 int이다.

`int main {`

.....

`} main ()` 항목에 대해서만 생각 가능

III, #include <iostream> : C++ 소스 파일을 컴파일하기 전에 <iostream>

헤더 파일을 읽어 C++ 소스 파일 안에 삽입자시

· <iostream> 헤더 파일에는 C++ 표준 입출력을 위한 클래스와 객체가

함수들이 있어 키보드 입력이나 화면 출력이 가능

### IV, 화면 출력

cout 기호 : · 스크린 창화와 연결된 C++ 표준 출력 스트림 기호

· 자신과 연결된 화면에 대신 출력

<< 연산자 : 스트림 삽입 연산자, 오른쪽 연산자 데이터를 왼쪽 스트림

기호에 삽입

V, 줄 끌기기 : '\n' or 'endl'

### 2. namespace 와 std::

I, namespace : 개별자와 자신만의 고유한 이름 공간을 생성

II, std:: : 표준 이름 공간, C++ 표준 라이브러리

### III, std:: 생략

1, using std:: ; 자식에 아래 모든 코드에서 std:: 사용

2, using namespace std:: ; std 이름 공간에 선언된 모든 이름에 아래 std:: 생략

### 3. 연산자

#### I, 산술 연산자

구분	연산자	의미	수학적 표현	C++ 표현	C++ 결과
단항 산술 연산자	+	양수	+3	+ 3	3
	-	음수	-2(-2)	-2(-2)	4
	+	덧셈	3+2	3+2	5
	-	뺄셈	10-2	10-2	8
이항 산술 연산자	*	곱셈	xy 또는 x×y	x*y	8
	/	나눗셈	x/y 또는 x÷y	x/y	4
	%	나머지	x mod y	x%y	1

#### II, 논리 연산자

AND → &&, OR → ||, NOT → !

12171398 신건희

### III. 관계연산자

· 두 개의 수 비교, 결과는 항상 T or F, 주로 for문이나 while문 등의  
제어문에서 조건을 검사하는 식으로 많이 사용

구분	연산자	의미	수학적 기호	C++ 표현	C++ 결과
비교 연산자	==	좌측이 우측과 같다	=	2 == 4	false
	!=	좌측이 우측과 같지 않다	≠	2 != 4	true
	>	좌측이 우측보다 크다	>	3 > 2	true
	>=	좌측이 우측보다 크거나 같다	≥	2 >= 3	false
	<	좌측이 우측보다 작다	<	4 < 5	true
	<=	좌측이 우측보다 작거나 같다	≤	4 <= 4	true

### IV, 대입연산자

연산자	사용 예	같은 의미의 수식
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
&=	x &= y	x = x & y
=	x  = y	x = x   y
^=	x ^= y	x = x ^ y
>>=	x >>= y	x = x >> y
<<=	x <<= y	x = x << y

### V, 비트 단위 연산자

연산자	명칭	설명
&	비트 논리곱(AND)	두 피연산자의 대응비트 두 비트가 모두 1이면 결과 비트는 1이 된다.
	비트 논리합(OR)	두 피연산자의 대응비트 두 비트 중 적어도 한 비트가 1이면 결과 비트는 1이 된다.
^	비트 베타적논리합(XOR)	두 피연산자의 대응비트 두 비트 중 1비트가 10번(두 비트가 서로 다른 비트일 때) 결과 비트는 1이 된다.
<<	왼쪽 이동(left shift)	두 번째 피연산자의 자릿개수만큼 첫번째 피연산자의 비트들을 왼쪽으로 이동시킨다. 오른쪽에 값 세우는 방법은 시스템에 따라 다르다.
>>	오른쪽 이동(right shift)	두 번째 피연산자의 자릿개수만큼 첫번째 피연산자의 비트들을 오른쪽으로 이동한다. 원쪽에 값 세우는 방법은 시스템에 따라 다르다.
~	1의 보수(one's complement)	모든 0비트는 1이 되고 모든 1비트는 0이 된다.

### VI, 증감 연산자

연산자	형태	연산자 호출	연산 의미
++	++x	전위 증가 연산자	연산 전에 x값 증가
	x++	후위 증가 연산자	연산 후에 x값 증가
--	--x	전위 감소 연산자	연산 전에 x값 감소
	x--	후위 감소 연산자	연산 후에 x값 감소

### VII, 정수형과 실수형

#### I, 정수형

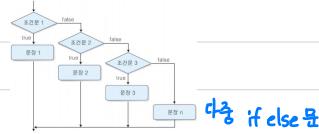
타입	최소값	최대값	크기(bytes)
signed short	-32768	32767	2
unsigned short	0	65535	2
signed int	-2147483648	2147483647	4
unsigned int	0	4294967295	4
signed long	-2147483648	2147483647	4
unsigned long	0	4294967295	4

#### II, 실수형

유형	크기	유효범위
float	4바이트	$\pm 3.4 \times 10^{-38} \sim \pm 3.4 \times 10^{38}$
double	8바이트	$\pm 1.7 \times 10^{-308} \sim \pm 1.7 \times 10^{308}$

## 5. 제어문

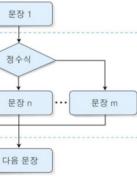
I, if 문, ifelse 문, 다중 if~else 문



다중 if else 문

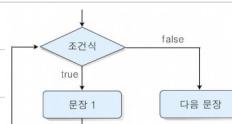
II, 다중선택 switch 문

```
switch(정수식) {
    case 정수값1 : 문장 1;[break];
    case 정수값2 : 문장 2;[break];
    ...
    case 정수값n : 문장n;[break];
    [default:] 문장n+1;
}
```



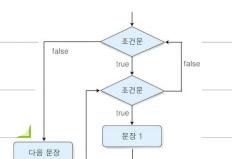
III, for 문 : 지정된 횟수 만큼 반복한다.

```
for(<초기식>;<조건식>;<증감식>)
    문장1;
}
```



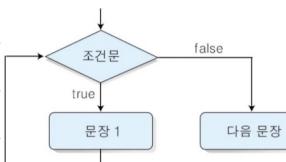
다중 for 문: for 문 안에 for 문을 포함, 서로 다른 제어 변수 사용

```
for(<초기식>;<조건식>;<증감식>)
    문장1;
    for(<초기식>;<조건식>;<증감식>)
        문장2;
    }
    문장3;
}
```



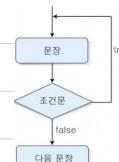
IV, While 문 : '...하는동안' 즉, 조건이 만족하는 동안 문장을 반복

**while(조건식){**  
 문장;  
**}**



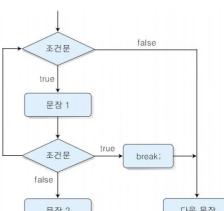
do ~ while 문: 조건이 거짓이 되도 한번은 실행

**do {**  
 문장  
**} while(조건식);**



V, break 문 : 프로그램의 일부를 수행하지 않고 멈춰놓고, 제어를 뛰어넘기 위해 사용

```
while(조건문) {
    문장 1;
    if(조건식)
        break;
    문장 2;
}
다음 문장;
```



## 6. 기억크래스터

기억클래스	유효범위(scope)	생존기간	메모리	초기화 여부
지역	블록 내	일시적	stack	쓰레기 값 있음 것 고려 중
전역	프로그램 내	영구적	메모리	숫자 0
지역	내부 : 블록 내 외부 : 모듈 내	영구적	메모리	숫자 0
지역	블록 내	일시적	CPU 내의 레지스터	쓰레기 값

\* static, auto 차이 살펴보기

```
01 #include<iostream>
02 using namespace std;
03 void sub();
04 void main()
05 {
06     for(int i=1; i<=5; i++){
07         cout<< i << "=====\n";
08         sub();
09     }
10 }
```

```
11 void sub()
12 {
13     cout<< "===== \n";
14     static int a = 0;
15     static int b = 0;
16     a+=100;
17     b+=100;
18     cout<< "    auto a = "<<a<<endl;
19     cout<< "    static b = "<<b<<endl;
20 }
```

```
21 =====
22     auto a = 100
23     static b = 100
24 =====
25     auto a = 100
26     static b = 200
27 =====
28     auto a = 100
29     static b = 300
30 =====
31     auto a = 100
32     static b = 400
33 =====
34     auto a = 100
35     static b = 500
Press any key to continue
```

int data = 100;

int \*print;

print = &data;

## 7. 포인터 변수

I, 주소에 의한 전달

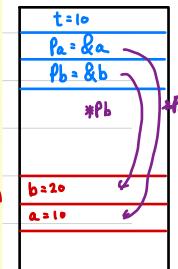
• 포인터 변수: 변수의 주소를 저장 [자료형은 변수명]

ex, t = \*pa : t에 pa의 주소 저장

주소를 얻어오기 위해 & 연산자를 변수 앞에 붙여

주소에 해당하는 값을 불러오면 \*연산자 사용

```
01 #include<iostream>
02 using namespace std;
03 void swap(int *pa, int *pb);
04 void main()
05 {
06     int a=10, b=20;
07     cout<< a >> " << a <<" b => " << b << "\n";
08     swap(&a, &b);
09     cout<< a >> " << a <<" b => " << b << "\n";
10 }
11 void swap(int *pa, int *pb)
12 {
13     int t;
14     t=*pa;
15     *pa=*pb;
16     *pb=t;
17 }
```



## 8. 배열

I, 특징: • 배열명만 가질 때 배열의 시작 주소의 값이라 이거

• 배열의 시작 주소만 알면 배열의 모든 원소의 값 알 수 있다.

II, 이차원 배열 (컴퓨터는 행 중심 배열)

배열 원소		배열 원소		배열 원소	
주소 값의 표현	저장	주소 값의 표현	저장	주소 값의 표현	저장
주소값 tAry[0][0]		tAry[0][1]		tAry[0][2]	
[12FF68] &tAry[0][0] tAry[0] *tAry		[12FF6C] &tAry[0][1] tAry[0] + 1 *tAry + 1	20	[12FF70] &tAry[0][2] tAry[0] + 2 *tAry + 2	12
tAry[1][0]		tAry[1][1]		tAry[1][2]	
[12FF74] &tAry[1][0] tAry[1] *tAry + 3 *(tAry+1)	3	[12FF78] &tAry[1][1] tAry[1] + 1 *tAry + 4 *(tAry+1)+1	5	[12FF7C] &tAry[1][2] tAry[1] + 2 *tAry + 5 *(tAry+1)+2	16

```
01 #include<iostream>
02 using namespace std;
03 void main()
04 {
05     int a[5] = {10,20,30,40,50};
06     cout<< a[0] >> " << a[0]>> " << a[0]<< "\n";
07     cout<< a[1] >> " << a[1]>> " << a[1]<< "\n";
08     cout<< a[2] >> " << a[2]>> " << a[2]<< "\n";
09     cout<< a[3] >> " << a[3]>> " << a[3]<< "\n";
10     cout<< a[4] >> " << a[4]>> " << a[4]<< "\n";
11 }
```

## 9. 구조체

### I. 예약어 struct 사용

II. 구조체의 멤버 참조를 위한 연산자 : dot(.) or arrow(→)

III. 함수의 매개변수를 구조체로 선언하여 구조체 인자를 값 전달

```
01 #include<iostream>
02 using namespace std;
03 struct namecard; // 구조체 선언
04 char name[20];
05 char job[30];
06 char tel[20];
07 char email[40];
08 };
09
10 void structPrn(namecard temp);
11 namecard structInput();
12
13 void main()
14 {
15     namecard x, y, z;
16
17     x=structInput();
18     y=structInput();
19     z=structInput();
20
21     cout<<"\n 이름    직업      연락처      이메일 ";
22     cout<<"\n=====";
23     structPrn(x);
24     structPrn(y);
25     structPrn(z);
26     cout<<"\n=====\n";
27 }
28
29 void structPrn(namecard temp) // 함수의 정의
30 {
31     cout<<"\n" <<temp.name <<"\t" <<temp.job
32         <<"\t" <<temp.tel <<"\t" <<temp.email;
33 }
34
35 namecard structInput() // 함수 정의
36 {
37     namecard temp;
38     cout<<"\n이름을 입력하세요=>";
39     cin>>temp.name;
40     cout<<"직업을 입력하세요=>";
41     cin>>temp.job;
42     cout<<"연락처를 입력하세요=>";
43     cin>>temp.tel;
44     cout<<"이메일을 입력하세요=>";
45     cin>>temp.email;
46
47     return temp;
48 }
```

```
이름을 입력하세요=>유지민
직업을 입력하세요=>웹마스터
연락처를 입력하세요=>453-0875
이메일을 입력하세요=>imhan@ide.com

이름을 입력하세요=>이재현
직업을 입력하세요=>웹디자이너
연락처를 입력하세요=>688-0885
이메일을 입력하세요=>eoh@ide.com

이름을 입력하세요=>나은경
직업을 입력하세요=>웹디자이너
연락처를 입력하세요=>789-0888
이메일을 입력하세요=>nwo@ide.com

이름    직업      연락처      이메일
=====

유지민 웹마스터 453-0875 imhan@ide.com
이재현 웹디자이너 688-0885 eoh@ide.com
나은경 웹디자이너 789-0888 nwo@ide.com

Press any key to continue
```

## 3장. 클래스와 객체

### 1. 클래스와 객체의 정의

클래스

I, 객체를 만들어내기 위해 정의된 설계도, 설계도

II, 멤버 변수와 멤버 함수 선언

객체

I, 클래스의 모양을 그대로 가지고 탄생 (붕어빵-틀), 설계도

II, 멤버 변수와 멤버 함수로 구성

III, 예외개의 객체 생성 가능, 각자 별도의 공간에 존재.

IV, 외부와의 정보 교환 및 통신을 위해 객체의 일부분 공개

### 2. 클래스 만들기

#### I, 클래스 선언부

- class 키워드 이용
- 멤버 변수와 함수 선언 : 클래스 선언 내 변수 초기화 X
- 멤버에 대한 접근 권한 지정 : private, public, protected 등의 허락
- 디폴트는 private

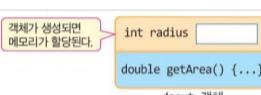
#### II, 클래스 구현부

```
double Circle::getArea() {
    return 3.14 * radius * radius;
}
```

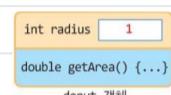
- 클래스가 정의된 모든 멤버 함수 구현

### 3. 객체 생성 및 활용

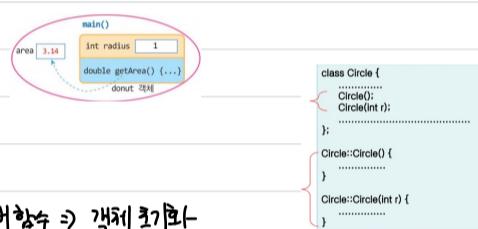
I, 객체 생성 : Circle donut;



II, 객체의 멤버 변수 접근 : donut.radius = 1;



III, 객체의 멤버 함수 접근 : double area = donut.getArea();



### 4. 생성자

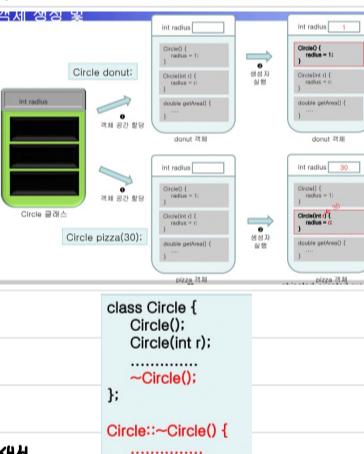
• 객체가 생성되는 시점에서 자동으로 호출되는 멤버 함수 => 객체 초기화

• 클래스 이름과 동일한 멤버 함수

• 리턴X, void 또한X

• 중복 가능, but 매개변수 type이나 개수가 달라야 함

• 선언되어 있지 않으면 기본 생성자(매개 변수X) 자동으로 생성



### 5 소멸자

• 객체가 소멸되는 시점에서 자동으로 호출

• 오직 한 번만 호출, 매개변수X

선언되어 있지 않으면 기본 소멸자(이루지도 하지 않고 단순 리턴) 자동 생성

• 생성된 역순으로 소멸

### 6. 인라인 함수

I, concept : inline 키워드로 선언된 함수

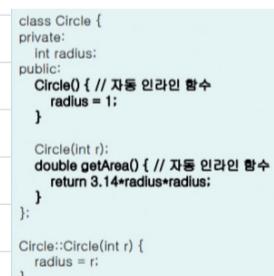
• 함수 호출에 따른 overhead X

• 실행 속도 ↑

II, 자동인라인 함수 : 클래스 선언부에 구현된 멤버함수

• inline 키워드 선언할 필요 X (자동으로 인라인 처리)

• 생성자 포함 모든 함수 가능



### 7. 구조체

• 언어와의 호환성 때문에 사용

• 클래스와 동일, but 디폴트 접근 지정은 public

• 구조체 객체 생성시 키워드 생략

```
struct StructName {
    private:
        // private 멤버 선언
    protected:
        // protected 멤버 선언
    public:
        // public 멤버 선언
};

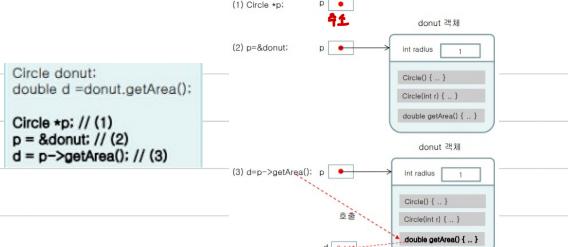
structName stObj; // (0) C++ 구조체 객체 생성
struct structName stObj; // (X) C 언어의 구조체 객체 생성
```

## 4장. 객체 배열, 동적 생성

### 1. 객체 포인터

• 객체의 주소값을 갖는 변수

• 포인터로 멤버를 접근 : 객체 포인터 → 멤버



### 2. 객체 배열

• 기본 타입 배열 선언과 형식 동일 : circle c[3]

• 매개변수 없는 생성자만 호출

Circle circleArray[3]; // 오류

• 초기화 방법 : 초기화 멤버 함수 선언, ex. setRadius()

• 배열의 각 원소 객체당 생성자 차집

```
Circle circleArray[3] = { Circle(10), Circle(20), Circle() };
```

### 3. 동적 메모리 할당 (new/delete)

• 실행 중에 운영체제로부터 할당

• 형식 : 데이터 타입 \*포인터 변수 = new 데이터 타입 [배열의 크기];

delete [] 포인터 변수;

• 배열의 동적 할당 형식 :



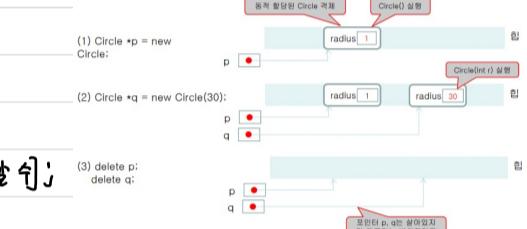
### 4. 객체와 객체 배열의 동적 생성 및 반환

#### I, 객체의 동적 생성 형식

클래스 이름 + 포인터 변수 = new 클래스 이름;

클래스 이름 + 쇄인터 변수 = new 클래스 이름 [매개변수를 놓아갈 수];

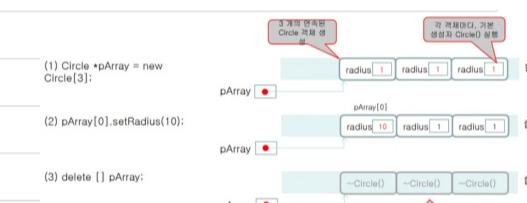
delete 포인터 변수;



#### II, 객체 배열의 ''

클래스 이름 + 포인터 변수 = new 클래스 이름 [배열 크기];

delete [] 포인터 변수;



### 5. this 포인터

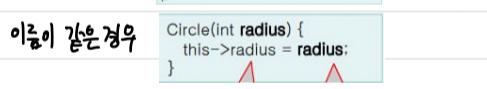
I, concepts : • 포인터, 객체 자신 포인터. 각 객체 속의 this는 다른 객체의 this와 다른

• 클래스의 멤버 함수 내에서만 사용



II, 필요 경우 : • 매개변수의 이름과 멤버 변수의 이름이 같은 경우

• 연산자 중복 시 필요



### 6. String 클래스

I, concept : • C++ 표준 라이브러리 ⇒ #include <string>

• 기본 기호의 문자열

• 다양한 문자열 연산 실행하는 연산자, 멤버 함수 포함

• 문자 위치 번호, 시작 번호 등 계산 중심, 0부터 시작임..

II, 동적 생성 :

```
string *p = new string("C++"); // 스트링 객체 동적 생성
cout << *p; // "C++" 출력
p->append(" Great!"); // p가 가리키는 스트링이 "C++ Great!" 이 됨
cout << *p; // "C++ Great!" 출력
delete p; // 스트링 객체 반납
```

#### III, 주요 연산자 & 멤버 함수

i, 문자열 연결

• a.append("문자") ; 뒤에 문자 붙이기

• +, += ; 그래서 연결

ii, 서브스트링

• b.substr(2,4) ; b의 2위치에서 4개의 문자 리턴

• b.substr(2) ; b의 2위치에서 끝까지 리턴

#### IV, 삽입

• a.insert(2,"really") ; 2의 2위치에 문자 삽입

• a.replace(2,11,"study") ; 2의 2위치부터 11개의 문자를 "study"로 대체

V, String to int : stoi;

• String year = '2014' j

int n = stoi(year) ; n은 2014

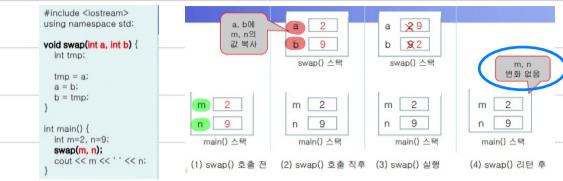
VI, getline(cin,a,'/n')

• a의 클래스에 enter 칠때 까지의 문자 입력

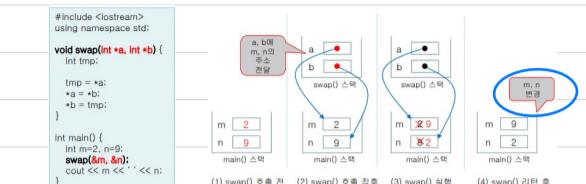
## CH5. 함수와 참조

### I. review

i) 값에 의한 호출 : 함수 호출 → 매개변수 스택생성, 실현과 순상X



ii) 주소에 의한 호출 : 매개변수는 포인터 타입, 호출하는 코드에서는 주소를 넘겨줌



### 1. 함수 호출 시 객체 전달

i) 값에 의한 호출

```
void increase(Circle c) {
    int r = c.getRadius();
    c.setRadius(r+1);
}

int main() {
    Circle waffle(30);
    increase(waffle);
    cout << waffle.getRadius() << endl;
}
```

• 호출하는쪽 (main)의 기능[!] → 매개변수 가변매개 변수  
• 생성자(X), 소멸자(X) → 비워짐  
⇒ 호출하는 순간의 실현과 적용은 매개변수 기준에 그대로 철달해짐  
만일 생성자/생성자연계의 반복은 0으로 초기화됨

ii) 주소에 의한 호출

```
void increase(Circle *p) {
    int r = p->getRadius();
    p->setRadius(r+1);
}

int main() {
    Circle waffle(30);
    increase(&waffle);
    cout << waffle.getRadius() << endl;
}
```

• 함수 (increase)의 매개변수는 객체에 대한 포인터 변수로 선언  
• 생성자(X), 소멸자(X) → 비워짐  
⇒ 호출하는 쪽의 주소가 포인터 p에 전달. P는 객체가 아닙니다.

### 3. 객체 치환, 대입

i) 치환

```
Circle c1(5);
Circle c2(30);
c1 = c2;
```

- 동일한 클래스 타입의 객체끼리 가능
- 두 객체는 내용만 같고獨立체인 공간까지

ii) 대입

```
Circle getCircle() {
    Circle tmp(30);
    return tmp; // 객체 tmp를 리턴한다.
}
```

• tmp를 리턴하고 소멸

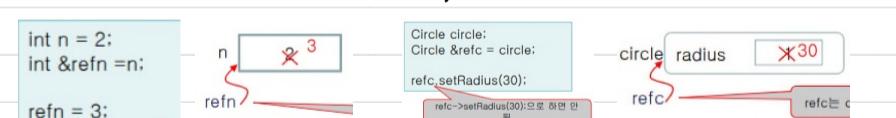
```
Circle c; // c의 반지름 1
c = getCircle(); // tmp 대입
```

### 4. 참조와 함수

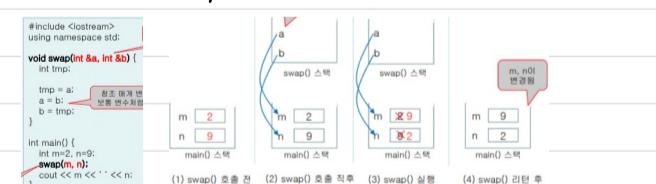
i) concepts

i), 참조자 & 도입

ii), 참조 변수는 이를만 생기며 새로운 공간 할당X, 기본 변수 공유



ii) 참조에 의한 호출 (call by reference)



- 변수 p, i는 스택에 생성되지만, 참조 매개변수 a, b는 이를만 관리하는 스택에 공간 할당X
- 참조 매개변수 이후엔 모든 영역은 원복 가능하여 대량 연산
- 생성자, 소멸자(X)

iii) 참조 리턴

- 혼란하는 공간에 대한 참조의 리턴.

```
char c = 'a';
char& find() { // char 타입의 참조 리턴
    return c; // 변수 c에 대한 참조 리턴
}

char a = find(); // a = 'a'가 됨
char &ref = find(); // ref는 c에 대한 참조
ref = 'M'; // c = 'M'

find() = 'b'; // c = 'b'가 됨
```

cf, char&, char\*, char 비교

char c = 'a' : 변수 C 생성, 'a'로 초기화  
char &p = &c : 포인터 변수 p 생성, p는 a의 주소 생성  
char &s = c : 변수 s는 이름만 생성, s는 a에 대한 별명

## CH7. 프렌드와 연산자 충돌

### I. C++ 프렌드

i) concepts : · private 멤버 사용을 허용해주는 함수

· 외부함수 (전역 함수, 다른 클래스의 멤버함수)

항목	프렌드 함수
존재	클래스 외부에 작성된 함수. 멤버가 아님
자격	클래스의 멤버 자격 부여. 클래스의 모든 멤버에 대한 접근 가능
선언	클래스 내에 friend 키워드로 선언
개수	프렌드 함수 개수에 제한 없음

### ii. 유형

i) 전역 함수 : 외부함수 equals를 Rect 클래스가 프렌드로 선언

```
class Rect { // Rect 클래스 선언
    ...
    friend bool equals(Rect r, Rect s);
};
```

ii) 다른 클래스의 멤버함수

```
class Rect { // 다른 클래스 선언
    ...
    friend bool RectManager::equals(Rect r, Rect s);
};
```

iii) 다른 클래스 전체

```
class Rect {
    ...
    friend RectManager;
};
```

### 2. 연산자 충돌

· (+) 언어에 본래부터 있는 연산자이 새로운 의미 지정

i) 특징 : 본래 있는 연산자만 충돌 가능

- 형식화 태이어 사용
- 연산자 함수 (operator function)을 이용하여 구현
- 반드시 클래스와 관계를 가짐

### ii) 연산자 함수

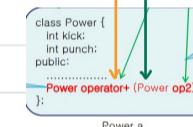
• 클래스의 멤버함수로 구현 or 외부함수로 구현하고 클래스에 프렌드 선언

```
Color operator+(Color op1, Color op2) {
    ...
    bool operator==(Color op1, Color op2) {
        ...
    }
}
```

```
class Color {
    ...
    Color operator+(Color op2);
    bool operator==(Color op2);
};
```

• 형식 : 리턴타입 operator 연산자 (매개변수) [멤버함수 중 하나]

ex, C = a+b; → C = a.operator+(b);



```
class Power {
    int kick;
    int punch;
public:
    ...
    Power operator+(Power op2);
}
```

+ 연산자 함수 코드

i) class 선언 후 j 불이기.

## CH7. 프렌드와 연산자 충복

### I. C++ 프렌드

- i, concepts : · private 멤버 사용을 허용해주는 함수  
· 외부함수 (전역 함수, 다른 클래스의 멤버함수)

항목	프렌드 함수
존재	클래스 외부에 작성된 함수, 멤버가 아님
자격	클래스의 멤버 자격 부여. 클래스의 모든 멤버에 대한 접근 가능
선언	클래스 내에 friend 키워드로 선언
개수	프렌드 함수 개수에 제한 없음

### II. 유형

- i, 전역 함수 : 외부함수 equals를 Rect 클래스의 프로토 타입으로 선언

```
class Rect { // Rect 클래스 선언
    ...
    friend bool equals(Rect r, Rect s);
};
```

- ii, 다른 클래스의 멤버 함수

```
class Rect {
    ...
    friend class RectManager; // 다른 클래스
    friend bool RectManager::equals(Rect r, Rect s);
};
```

- iii, 다른 클래스 정체

```
class Rect {
    ...
    friend class RectManager;
};
```

### 2. 연산자 충복

- (++) 연산자 본래부터 있는 연산자에 새로운 의미 부여

- i, 특징 : · 본래 있는 연산자만 충복 가능

- 표현식 태입이 가능

- 연산자 함수 (operator function) 을 이용하여 구현

- 반드시 클래스와 관계를 가짐

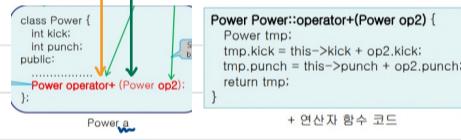
### 3. 이항연산자 충복

- 클래스의 멤버함수로 구현 or 외부함수로 구현하고 클래스에 프렌드 선언

```
Color operator + (Color op1, Color op2) {
    ...
    bool operator == (Color op1, Color op2) {
        ...
    }
}
class Color {
    ...
    Color operator + (Color op2);
    bool operator == (Color op2);
};
```

- 형식 : 리턴타입 operator 연산자 (매개변수) [ 멤버함수 충복 ]

ex,  $c = a + b;$   $\rightarrow c = a + (b);$



```
Power Power::operator+(Power op2) {
    Power tmp;
    tmp.kick = this->kick + op2.kick;
    tmp.punch = this->punch + op2.punch;
    return tmp;
}
```

+ 연산자 함수 코드

### 4. 단항연산자 충복

- i, 전위 ++ 연산자 충복

$++a \xrightarrow{\text{compile}} a.++()$   $b = ++a$  이고 show를 호출하여 a도 증가한 것으로 나옴

```
class Power {
    ...
    public:
        Power operator++ () {
            // 매개 변수 없음
        }
};
Power a
```

전위 ++ 연산자 함수 코드

- ii, 후위 ++연산자 충복

$a++ \xrightarrow{\text{compile}} a++ (\text{일의정수})$

$b = a++$  이고 show를 호출하여 a는 증가한 것이 나오고 b는 증가하지 않은 a

```
class Power {
    ...
    public:
        Power operator++ (int x) {
            // 매개 변수
        }
};
Power a
```

후위 ++ 연산자 함수 코드

### 5. 프렌드를 이용한 연산자 충복

- i,  $b = 2 + a;$   $\boxed{b = 2 + (a);}$  (X) : 는 가능하지 않다  
 $b = + (2, a);$

```
class Power {
    int kick;
    int punch;
    public:
        Power(int kick=0, int punch=0) {
            this->kick = kick; this->punch = punch;
        }
        void show();
        friend Power operator+(int op1, Power op2); // 프렌드 선언
};
```

```
Power operator+ (int op1, Power op2) {
    Power tmp;
    tmp.kick = op1 + op2.kick;
    tmp.punch = op1 + op2.punch;
    return tmp;
}
```

주의 :

- ii,  $c = a + b;$   $\rightarrow c = + (a, b);$

```
class Power {
    int kick;
    int punch;
    public:
        Power(int kick=0, int punch=0) {
            this->kick = kick; this->punch = punch;
        }
        void show();
        friend Power operator+(Power op1, Power op2); // 프렌드 선언
};
```

```
Power operator+ (Power op1, Power op2) {
    Power tmp;
    tmp.kick = op1.kick + op2.kick;
    tmp.punch = op1.punch + op2.punch;
    return tmp;
}
```

- iii,  $++a \rightarrow ++(a)$

```
Power operator++ (Power& op) {
    op.kick++;
    op.punch++;
    return op;
}
```

```
Power operator++ (Power& op, int x) {
    Power tmp = op;
    op.kick++;
    op.punch++;
    return tmp;
}
```

참고 안하면 오류는  
비주시 않음

## CH.8 상속

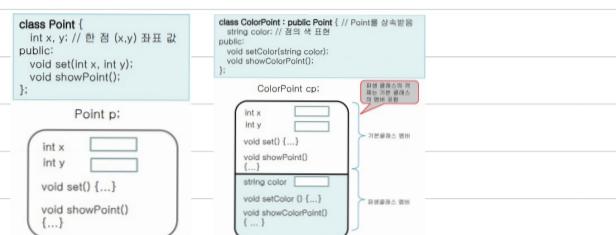
### I. concept

- 클래스 사이에서 상속 관계 정의 (개체 사이 X)
- 기본 클래스 → 파생 클래스
- 장점: 간결한 클래스 작성, 클래스 간의 재활용성 및 관리의 용이, 생산성 향상

### 2. 상속 선언

I, class Student : public Person

파생 클래스 명 상속 선언 기반 클래스명



### II. 접근

- 파생 클래스의 멤버들은 기본 클래스의 private 멤버에 접근 가능
- 기본 클래스의 private 멤버는 파생 클래스에 상속되고 파생 클래스의 개체에도 포함되지만 파생 클래스에서 접근 못함  
set() or showPoint()로 간접적으로 접근해야 함.

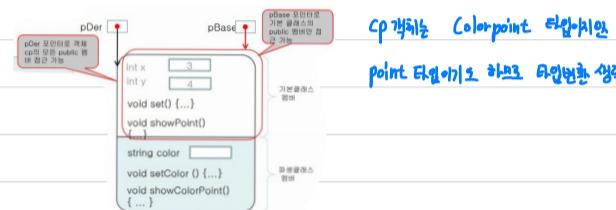
### 3. 개체 포인터

I, 엔캐스팅: 파생 클래스의 객체를 기본 클래스의 포인터로 가리킴

$$\text{Point } * \text{pBase} \text{ (기본 클래스 포인터)} = (\text{point } *) \text{ pDer} \text{ (파생 클래스의 주소)}$$

```
int main() {
    ColorPoint cp;
    ColorPoint *pBase = &cp;
    Point *pBase = pDer; // 엔캐스팅

    pDer->set(3,4);
    pBase->showPoint();
    pDer->setColor("Red");
    pDer->showColorPoint();
    pBase->showColorPoint(); // 경파일 오류
}
```



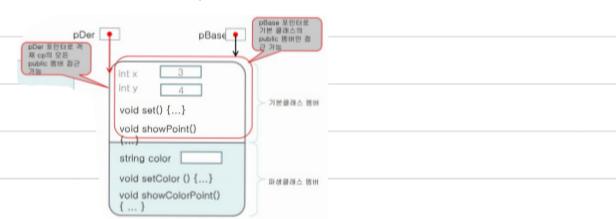
pBase 포인터로는 ColorPoint 라이브러리의 Point 타입이기고 차트로 내용변환 상당

II, 다운캐스팅: 기본 클래스 포인터가 가리키는 개체를 파생 클래스의 포인터로 가리키는 것.

$$\text{pDer (파생 클래스 포인터)} = (\text{colorPoint } *) \text{ pBase (기본 클래스 포인터)}$$

```
int main() {
    ColorPoint cp;
    ColorPoint *pDer;
    Point *pBase = &cp; // 엔캐스팅

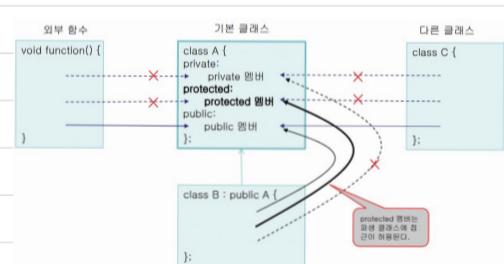
    pBase->showPoint();
    pDer->setColor("Red"); // 경파일 오류
    pDer->showColorPoint(); // 경파일 오류
}
```



### 4. protected 접근 제한자

I, private 멤버: 선언된 클래스 내에서만 접근 가능

· 파생 클래스에서 접근 X



II, public 멤버: 모두 접근 가능

III, protected 멤버: 선언된 클래스에서 접근 가능

· 파생 클래스에서만 접근 허용

### 5. 상속과 생성자 소멸자

I, 파생 클래스의 개체가 생성될 때 파생, 기본 클래스의 생성자 모두 생성되고 기본 클래스의 생성자가 먼저 생성

- 기본 클래스의 초기화가 먼저 이루어지고 이를 활용하는 파생 클래스의 초기화가 나중에 이루어짐
- 컴파일러는 파생 클래스의 개체 코드를 컴파일 할 때 기본 클래스의 개체 초기화 코드 삽입

II, 소멸자는 파생 → 기본 순서로 실행

III, 기본 클래스의 생성자 선택

```
class A {
public:
    A() { cout << "생성자 A" << endl; }
    ~A() { cout << "파괴자 A" << endl; }
};

class B : public A {
public:
    B() { cout << "생성자 B" << endl; }
    ~B() { cout << "파괴자 B" << endl; }
};

int main() {
    B b();
}
```

파생 클래스의 생성자가 명시적으로 기본 클래스의 생성자를 선택

명시적 선택 코드가 없으면 컴파일러는 기본 클래스의 기본생성자 호출되도록 무시적으로 선택

### 6. 상속의 종류

I, public 상속: 기본 클래스의 protected, public 멤버들을 접근 자정 없이 그대로 상속

II, protected 상속: "protected 접근 자정으로 변경되어" 상속

III, private 상속: "private" "

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

class Base {
    int a;
protected:
    void setA(int a) { this->a = a; }
public:
    void showA() { cout << a; }
};

class Derived : private Base {
    int b;
protected:
    void setB(int b) { this->b = b; }
public:
    void showB() { cout << b; }
};

int main() {
    Derived x;
    x.a = 5; // ①
    x.setA(10); // ②
    x.showA(); // ③
    x.b = 10; // ④
    x.setB(10); // ⑤
    x.showB(); // ⑥
}
```

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

class Base {
    int a;
protected:
    void setA(int a) { this->a = a; }
public:
    void showA() { cout << a; }
};

class Derived : protected Base {
    int b;
protected:
    void setB(int b) { this->b = b; }
public:
    void showB() { cout << b; }
};

int main() {
    Derived x;
    x.a = 5; // ①
    x.setA(10); // ②
    x.showA(); // ③
    x.b = 10; // ④
    x.setB(10); // ⑤
    x.showB(); // ⑥
}
```

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

class Base {
    int a;
protected:
    void setA(int a) { this->a = a; }
public:
    void showA() { cout << a; }
};

class Derived : private Base {
    int b;
protected:
    void setB(int b) { this->b = b; }
public:
    void showB() { cout << b; }
};

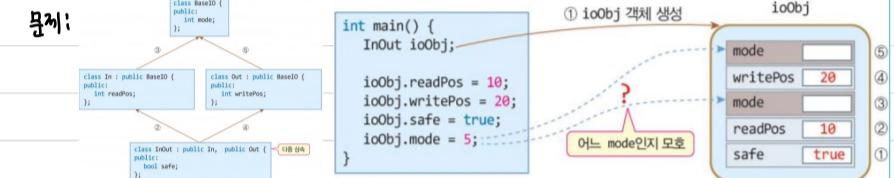
class GrandDerived : private Derived {
    int c;
protected:
    void setA(int x) { // ③
        setA(x); // ②
        showA(); // ④
        setB(x); // ⑤
    };
public:
    void showB() { cout << b; }
};

int main() {
    GrandDerived x;
    x.a = 5; // ①
    x.setA(10); // ②
    x.showA(); // ③
    x.b = 10; // ④
    x.setB(10); // ⑤
    x.showB(); // ⑥
}
```

7. 다중 상속: 하나의 파생 클래스가 여러 클래스를 동시에 상속 받음

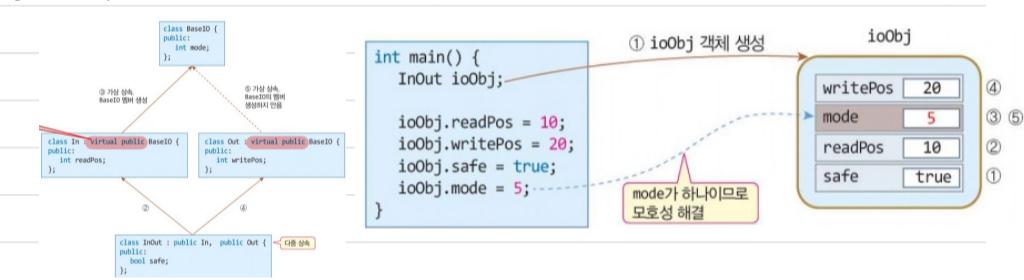
class Musicphon : public mp3, public Mobile phone

문제:



디아이 모드의 다중 상속 구조는 기본 클래스 멤버가 충돌되면서 경파일 오류 발생 → 기상상속으로 해결

### 8. 가상상속



· 파생 클래스의 개체가 생성될 때 기본 클래스 멤버는 오직 한번 생성



## CH.13 예외 처리, 링크리거

### I. 예외

#### I, 형식

```

n = 0
sum = 15
try {
    if(n == 0) { // 예외 발생 코드
        throw n;
    } else {
        average = sum / n;
        ...
    }
} catch(int x) { // 예외 처리 코드
    cout << "예외 발생!!";
    cout << x << "로 나눌 수 없음" << endl;
    average = 0;
}
cout << "평균 = " << average << endl;

```

예외 발생!! 0으로 나눌 수 없음  
평균 = 0

- try {} : 예외가 발생할 가능성이 있는 코드를 블록
- throw x : 예외 발생을 알림, try {} 내에서 선언되어야 함, 거의 반드시에 맞는 catch문과 함께
- catch(변수리거 변수) : throw에 의해 발생한 예외 처리
- catch 이후 되돌아 가지 않고 다음으로 넘기감
- 문자열 예외를 던지는 throw와 catch() {} 블록

#### 개념 II, catch(클래스형 &c)

#### 문자열 블록에 catch(char\* s)

- (catch(...)) : 거의 병행되기에 관계없이 모든 예외 처리, 그러나 throws는 구체화된 예외를 명시해줌

#### III, throw, catch의 예

##### 1. 하나의 try {}에 다수 catch() 연결

```

try {
    ...
    throw "음수 불가능";
    ...
    throw 3;
}
catch(char* s) { // 문자열 터전을 전달
    ...
}
catch(int x) { // int 타입 예외
    ...
}

```

##### 2. 함수를 포함하는 try {} 블록

```

int main() {
    int n = multiply(2, -3);
    cout << "곱은 " << n << endl;
    catch(char* negative) { // 함수 호출
        cout << "exception happened: " << negative;
    }
    ...
    else
        return x*y;
}

```

exception happened: 음수 불가능  
예외 던지기

#### IV, 예외를 발생시키는 함수 선언

```

int max(int x, int y) throw(int) {
    if(x < 0) throw x;
    else if(y < 0) throw y;
    else if(x > y) return x;
    else return y;
}

double valueAt(double *p, int index) throw(int, char*) {
    if(index < 0)
        throw "index out of bounds exception";
    else if(p == NULL)
        throw 0;
    else
        return p[index];
}

```

모두 int 타입 예외 발생  
char\* 타입 예외 발생  
int 타입 예외 발생

#### V, 함수 형성에 있어서 throw(예외 터칭, 예외 타입, ...)

- 각 등을 명확히 하고 가독성을 높임

#### VI, 함수 throw(int or double or char\* or ...)

#### VII, try {} 내에 try {} 블록의 중첩 가능

```

try {
    ...
    throw 3;
}
try {
    ...
    throw "abc";
    ...
    throw 5;
}
catch(int inner) {
    cout << inner; // 5 출력
}
catch(char* s) {
    cout << s; // "abc" 출력
}
catch(int outer) {
    cout << outer; // 3 출력
}

```

#### VIII, 예외 경우 종합

1. 하나의 try {}에 다수 catch() 연결
2. 외부 함수를 통해 throw 가능, 함수는 try {} 내에만 가능
3. try {} 내에 try {} 중첩 가능
4. catch() 내에서 다시 throw 가능
5. 예외는 함수를 여러개 전너서도 전달 가능
6. 받은 예외를 다시 던질 수 있음
7. try {} 내에 여러개 catch() 있을 수 있음
8. 구체적인 예외를 먼저 참조

#### IX, 예외 클래스 만들기 : throw로 객체를 던짐

```

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

class MyException { // 사용자가 만드는 기본 예외 클래스 선언
    int lineNumber;
    string func, msg;
public:
    MyException(int ln, string f, string m) {
        lineNumber = ln; func = f; msg = m;
    }

    void print() { cout << func << ":" << lineNumber << ", " << msg;
    << endl; }

    class DivideByZeroException : public MyException { // 0으로 나누는 예외 클래스 선언
    public:
        DivideByZeroException(int lineNumber, string func, string msg) : MyException(lineNumber, func, msg) {}
    };

    class InvalidInputException : public MyException { // 잘못된 입력 예외 클래스 선언
    public:
        InvalidInputException(int lineNumber, string func, string msg) : MyException(lineNumber, func, msg) {}
    };
}

```

```

int main() {
    int x, y;
    cout << "나눗셈을 합니다. 두 개의 양의 정수를 입력하세요>";
    cin >> x >> y;
    if(x < 0 || y < 0)
        throw InvalidInputException(32, "main()", "음수 입력 예외 발생");
    if(y == 0)
        throw DivideByZeroException(34, "main()", "0으로 나누는 예외 발생");
    cout << (double)x / (double)y;
}

catch(DivideByZeroException &e) {
    e.print();
}

catch(InvalidInputException &e) {
    e.print();
}


```

나눗셈을 합니다. 두 개의 양의 정수를 입력하세요>2 5  
0.4  
나눗셈을 합니다. 두 개의 양의 정수를 입력하세요>200 -3  
main():32, 음수 입력 예외 발생  
나눗셈을 합니다. 두 개의 양의 정수를 입력하세요>20  
main():34, 0으로 나누는 예외 발생

## CH.11 C++ 입출력 시스템

### I. 스트림

- 스트림은 프로그램과 장치를 연결하여 바이트 단위로 입출력한다.
- 입력스트림(ex, cin) : 입력 장치와 프로그램을 연결해주는 객체
- 출력스트림(ex, cout) : 출력 장치와 프로그램을 연결해주는 객체

#### II. ostream 클래스의 멤버함수를 이용한 문자 출력

- I, cout.put(char ch) : ch의 한 문자를 출력  
II, cout.write(char\* str, int n) : str에 있는 n개의 문자 출력

#### III. istream 클래스의 멤버함수를 이용한 문자 입력

- I, 문자가져 입력  
· int get() : 입력스트림에서 문자 읽어 리턴  
· <Enter> 키 등의 공백문자도 읽어 리턴된다.

```

int ch;
while((ch = cin.get()) != EOF) { // EOF는 -1
    cout.put(ch); // 읽은 문자 출력
    if(ch == '\n')
        break; // <Enter> 키가 입력되면 읽기 종단
}

```

while 문의 cin.get()은 키 입력을 대체하기 사용하거나 <Enter> 키를 일정하면 읽기 종단

- get(char &ch) : 입력스트림에서 문자 읽어 ch에 저장하고 리턴  
· <Enter> 키 등의 공백문자도 읽어 리턴된다.

```

char ch;
cin.get(true);
while(ch != EOF) {
    ch = cin.get(); // 입력된 키를 ch에 저장하여 리턴
    if(cin.eof())
        break; // EOF를 만나면 읽기 종료
    cout.put(ch); // ch의 문자 출력
    if(ch == '\n')
        break; // <Enter> 키가 입력되면 읽기 종단
}

```

#### IV, 문자 여러개 입력

- get(char\* s, int n) : n-1개의 문자를 읽어 배열 s에 저장하고 마지막에 '\0' 삽입  
char str[10]; // '\0'을 만면 '\0'을 삽입하고 리턴 => '\0'은 스트림 버퍼에 남아있음  
cin.get(str, 10); // s => cin.get() or cin.ignore()을 통해 '\0' 처리

- getline(char\* s, int n, char delm='\'\n\') : get()과 동일하지만 delm이 사용된 문자를 스트링에서 처리

· delm 마지막 문자가 생략되면 delm=<Enter>

#### V, 입력 문자 건너뛰기 : ignore(int n=1, int delm=EOF) : n개의 문자 건너뛰기, 중간에 delm 만나면 계산을 중단

- 최근에 읽은 문자 개수 리턴 : int gcount()

```

char line[80];
cin.getline(line, 80);
int n = cin.gcount(); // 최근의 실행한 getline() 함수에서 읽은 문자의 개수 리턴

```

#### VI. 입출력 format(형식) 지정 방법

##### I, format 플래그

- 하나의 풀리는 한 번로 표현도며 한 가지 형식 정보를 표현

· cin, cout은 이 풀리기 세팅된 풀리고 값을 반영하여 포맷 입출력을 수행

long setf(long flags)	flags를 스트림의 포맷 플래그로 설정. 이전 플래그 값 리턴
long unsetf(long flags)	flags에 설정된 비트 값에 따라 스트림의 포맷 플래그 해제. 이전 플래그 값 리턴

cout.unsetf(ios::dec); // 10진수 해제  
cout.setf(ios::hex); // 16진수로 설정  
cout << 30 << endl; // 1e8 출력됨

#### II, format 함수

##### << 연산자를 이용한 예제

int width(int minWidth)	출력되는 필드의 최소 너비를 minWidth로 설정. 이전에 설정된 너비 값 리턴
char fill(char cFill)	필드의 빈 칸을 cFill 문자로 채우도록 지정. 이전 문자 값 리턴
int precision(int np)	출력되는 수의 유효 숫자 자릿수 np 개로 설정. 정수 부분과 소수점(.)은 제외

cout.fill(' ');  
cout.width(10);  
cout << "Hello" << endl;

^^^^^ Hello

#### III, 조작자

- << or >> 연산자와 함께 사용

i, 매개 변수 없는 조작자

```

cout << hex << showbase << 30 << endl;
cout << dec << showpos << 100 << endl;

```

0x1e  
+100

ii, 매개 변수 있는 조작자 : #include <iomanip> 필요

```

cout << setw(10) << setfill(' ') << "Hello" << endl;

```

^^^^^Hello