Lab 7

목차

- Destructor
- Copy Constructor

- Macro
- Union

https://padlet.com/dcslabcp/0426-dzhnbgt9gp1wjvvn

Destructors

- 객체가 소멸될 때 자동으로 실행되는 클래스의 멤버 함수
  - 프로그램의 종료, scope의 끝, 삭제 키워드를 이용한 명시적 delete

- 해당 객체가 lifetime 동안 사용했던 자원을 풀어주는 청소의 역할

- 소멸자의 규칙
  - 소멸자의 이름은 클래스 이름과 동일하며, 앞에 ~ 를 달아서 표시
  - 인수가 없다
  - 반환값이 없다

Destructors

- 클래스의 모든 non-static member의 destructor는 constructor와 반대 순서로 실행된다

```cpp
#include <iostream>
class A{
  public:
    A() {
      std::cout << "Constructor(A)" << std::endl;
    }
    ~A() {
      std::cout << "Destructor(A)" << std::endl;
    }
};
class B {
  public:
    B() {
      std::cout << "Constructor(B)" << std::endl;
    }
    ~B() {
      std::cout << "Destructor(B)" << std::endl;
    }
};
```

```cpp
class C{
  A a;
  B b;
};

int main() {
  C c;
}
```

```
Constructor(A)
Constructor(B)
Destructor(B)
Destructor(A)
```

Destructors (1)

```cpp
#include <iostream>
class Ex {
  public:
    int i;
    Ex(int i): i(i) {
      std::cout << "constructor" << i << std::endl;
    }
    ~Ex() {
      std::cout << "destructor" << i << std::endl;
    }
};
```

```cpp
Ex ex0(0);
int main() {
  Ex ex1(1);
  Ex* p_ex;
  {
    Ex ex2(2);
    p_ex = new Ex(3);
  }
  delete p_ex;
}
```

출력 결과가 어떻게 될까요 ?

Destructors (1)

```
constructor0
constructor1
constructor2
constructor3
destructor2
destructor3
destructor1
destructor0
```

```
Ex ex0(0); //constructor0
int main() {
    Ex ex1(1); //constructor1
    Ex* p_ex;
    {
        Ex ex2(2); //constructor2
        p_ex = new Ex(3); //constructor3
    } //destructor2
    delete p_ex; //destructor3
} //destructor1
//destructor0
```

Destructors (2)

```cpp
#include <iostream>
struct Base {
  std::string base_string;
  Base(std::string val) : base_string(val){
    std::cout << "Base : " << val << "\n";
  }
  virtual ~Base() {
    std::cout << "~Base : " << base_string << "\n";
  }
};

struct Derived : public Base {
  std::string derived_string;
  Derived(std::string val) : Base(val), derived_string(val) {
    std::cout << "Derived : " << val << "\n";
  }
  virtual ~Derived(){
    std::cout << "~Derived : " << derived_string << "\n";
  }
};
```

```cpp
int main() {
  Base thing1("thing1");
  Derived* p;
  {
    p = new Derived("p_inside");
    Derived inner_thing("inside");
  }
  Derived thing2("thing2");
  delete p;
}
```

출력 결과가 어떻게 될까요 ?

Destructors (2)

```
Base : thing1
Base : p_inside
Derived : p_inside
Base : inside
Derived : inside
~Derived : inside
~Base : inside
Base : thing2
Derived : thing2
~Derived : p_inside
~Base : p_inside
~Derived : thing2
~Base : thing2
~Base : thing1
```

```cpp
int main() {
  Base thing1("thing1"); // Base : thing1
  Derived* p;
  {
    p = new Derived("p_inside"); // Base : p_inside, Derived : p_inside
    Derived inner_thing("inside"); // Base : inside, Derived : inside
  } // ~Derived : inside, ~Base : inside
  Derived thing2("thing2"); //Base : thing2, Derived : thing2
  delete p; //~Derived : p_inside, ~Base : p_inside
} // ~Derived : thing2, ~Base : thing2
// ~Base : thing1
```

Copy Constructor

- 이미 만들어진 같은 클래스의 다른 객체를 사용하는 생성자
- 컴파일러가 기본적으로default copy constructor 제공함

```
#include <iostream>
class C{
  public:
    int n;
    C(int n = 1): n(n) { }
    C(const C& c) : n(c.n) { }
};

int main(){
  C c1(10);
  C c2(c1);
  std::cout << c2.n <<std::endl; //10
}
```

Copy Constructor

- Copy constructor가 호출되는 경우
  - Initialization

    C c2(c1); 또는 C c3= c1; c1이 C 객체인 경우
  - Function argument passing by value

    F(c1); c1이 C 객체이고, f가 void f(C c) 인 경우
  - Function return by value

    C f() 에서return a가 호출되는 경우 a가 C객체인 경우

Copy Constructor

```cpp
#include <iostream>
class C{
  public:
    int n;
    C(int n = 1): n(n) { }
    C(const C& c) : n(c.n) {
      std::cout << "copy con" << std::endl;
    }
};

void f_get_C(C c){
  std::cout << "function f" << std::endl;
}

C f_return_C() {
  return C();
}
```

```cpp
int main(){
  C c1(10);
  C c2(c1); //"copy con"
  C c3 = c1; //"copy con"
  f_get_C(c1); //"copy con", "function f"
  f_return_C(); //"copy con"
}
```

Copy elision

(since C++11) constructors

- C++ 컴파일러가 불필요한 객체 복사를 하지 않도록 최적화
- Return value optimization (RVO) : 함수의 리턴값을 저장하는 임시 객체를 컴파일러가 제거

```
#include <iostream>

struct C {
  C() = default;
  C(const C&) { std::cout << "Copycon\n"; }
};

C f() {
  return C();
}

int main() {
  std::cout << "Hello World!\n";
  C obj = f();
}
```

1.C default 생성자 생성

2.같은 C default 생성자 생성하여 C obj에 대입

3. 같은 C default 라는것을 컴파일러가 파악 생략함 => copy 생성자 생략

```
Hello World!
Copycon
Copycon
```

```
Hello World!
Copycon
```

```
Hello World!
```

Implicit Copy Constructor

- Copy constructor를 따로 정의하지 않는 경우, 기본적으로 제공
- pointer-type attribute의 경우 shallow copy

```cpp
#include <string>
#include <iostream>

class Artist {
  public:
    std::string name;
    Artist(std::string name): name(name) { }
    void change_name(const std::string str){
      name = str;
    }
};

class Song {
  public:
    Artist* artist;
    Song(Artist* artist): artist(artist) { }
};
```

```cpp
int main(){
  Artist* art1 = new Artist("IU");
  Song rollin(art1);
  Song lilac(rollin);
  rollin.artist->change_name("Brave girls");
  std::cout << rollin.artist->name << std::endl;
  std::cout << lilac.artist->name << std::endl;
}
```

출력 결과가 어떻게 될까요 ?

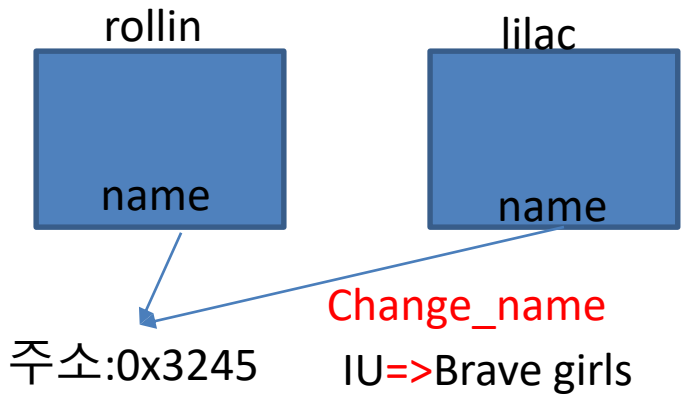
Implicit Copy Constructor – shallow copy

```
int main(){
  Artist* art1 = new Artist("IU");
  Song rollin(art1);
  Song lilac(rollin);
  rollin.artist->change_name("Brave girls");
  std::cout << rollin.artist->name << std::endl;
  std::cout << lilac.artist->name << std::endl;
}
```

```
Brave girls
Brave girls
```

rollin과 lilac 의 artist가 같은 art1을 가리키고 있고, art1의 name이 Brave girls 로 바뀌기 때문입니다

출력결과로 Brave girls IU를얻으려면어떻게해야할까요?

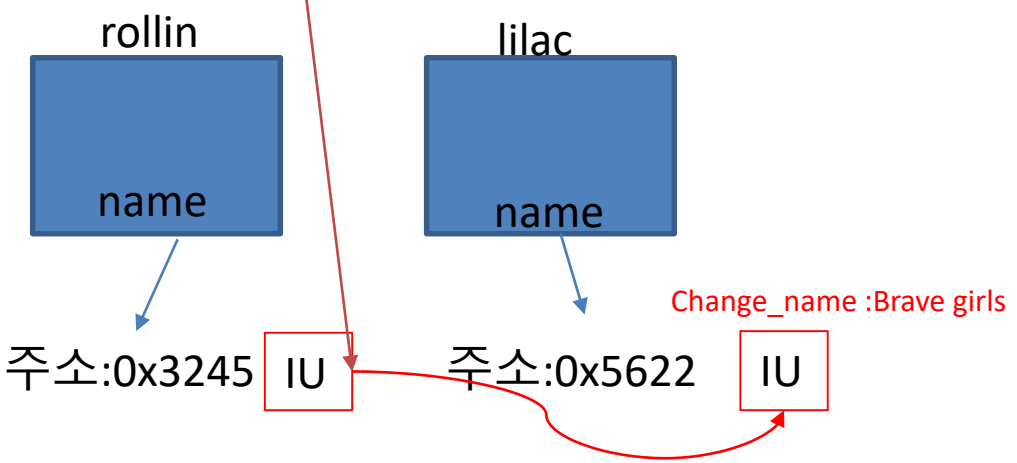


실습: Copy Constructor – deep copy

```cpp
#include <string>
#include <iostream>

class Artist {
  public:
    std::string name;
    Artist(std::string name): name(name) { }
    Artist(Artist const &artist): name(artist.name) { }
    void change_name(const std::string str){
      name = str;
    }
};

class Song {
  public:
    Artist* artist;
    Song(Artist* artist): artist(artist) { }
    //복사 생성자에 만들어보기
};
```

```cpp
int main(){
  Artist* art1 = new Artist("IU");
  Song rollin(art1);
  Song lilac(rollin);
  rollin.artist->change_name("Brave girls");
  std::cout << rollin.artist->name << std::endl;
  std::cout << lilac.artist->name << std::endl;
}
```

```
Brave girls
IU
```



Default 생성자의 한계

포토캐논 겹치기 예제

```
// 디폴트 복사 생성자의 한계
#include <string.h>
#include <iostream>

class Photon_Cannon {
  int hp, shield;
  int coord_x, coord_y;
  int damage;

  char *name;

 public:
  Photon_Cannon(int x, int y);
  Photon_Cannon(int x, int y, const char *cannon_name);
  ~Photon_Cannon();

  void show_status();
};
```

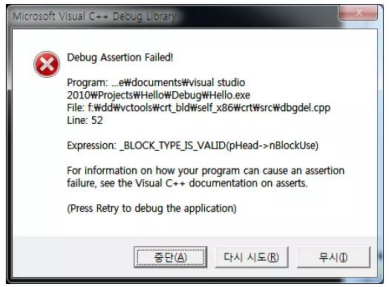
```
Photon_Cannon::Photon_Cannon(int x, int y, const char *cannon_name) {
  hp = shield = 100;
  coord_x = x;
  coord_y = y;
  damage = 20;

  name = new char[strlen(cannon_name) + 1];
  strcpy(name, cannon_name);
}
Photon_Cannon::~Photon_Cannon() {
  // 0 이 아닌 값은 if 문에서 true 로 처리되므로
  // 0 인가 아닌가를 비교할 때 그냥 if(name) 하면
  // if(name != 0) 과 동일한 의미를 가질 수 있다.

  // 참고로 if 문 다음에 문장이 1 개만 온다면
  // 중괄호를 생략 가능하다.

  if (name) delete[] name;
}
void Photon_Cannon::show_status() {
  std::cout << "Photon Cannon :: " << name << std::endl;
  std::cout << " Location : ( " << coord_x << " , " << coord_y << " ) "
            << std::endl;
  std::cout << " HP : " << hp << std::endl;
}
int main() {
  Photon_Cannon pc1(3, 3, "Cannon");
  Photon_Cannon pc2 = pc1;

  pc1.show_status();
  pc2.show_status();
}
```

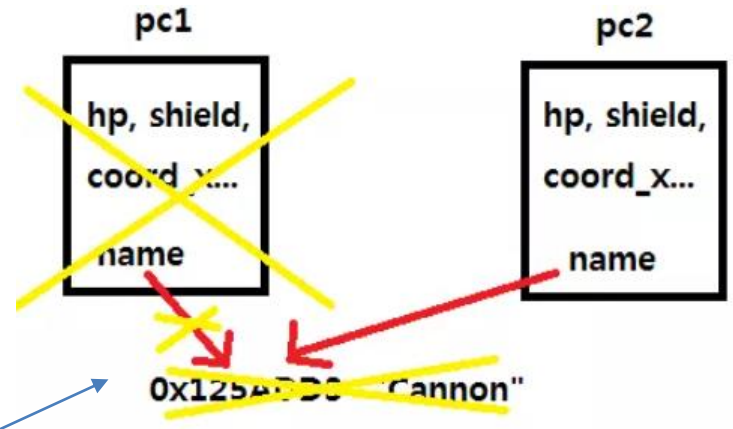
Default 생성자의 한계

```
Photon_Cannon::Photon_Cannon(int x, int y, const char *cannon_name) {
  hp = shield = 100;
  coord_x = x;
  coord_y = y;
  damage = 20;

  name = new char[strlen(cannon_name) + 1];
  strcpy(name, cannon_name);
}
Photon_Cannon::~Photon_Cannon() {
  // 0 이 아닌 값은 if 문에서 true 로 처리되므로
  // 0 인가 아닌가를 비교할 때 그냥 if(name) 하면
  // if(name != 0) 과 동일한 의미를 가질 수 있다.

  // 참고로 if 문 다음에 문장이 1 개만 온다면
  // 중괄호를 생략 가능하다.

  if (name) delete[] name;
}
void Photon_Cannon::show_status() {
  std::cout << "Photon Cannon :: " << name << std::endl;
  std::cout << " Location : ( " << coord_x << " , " << coord_y << " ) "
            << std::endl;
  std::cout << " HP : " << hp << std::endl;
}
int main() {
  Photon_Cannon pc1(3, 3, "Cannon");
  Photon_Cannon pc2 = pc1;

  pc1.show_status();
  pc2.show_status();
}
```

```
Photon_Cannon::Photon_Cannon(const Photon_Cannon& pc) {
  hp = pc.hp;
  shield = pc.shield;
  coord_x = pc.coord_x;
  coord_y = pc.coord_y;
  damage = pc.damage;
  name = pc.name;
}
```

Default 생성자



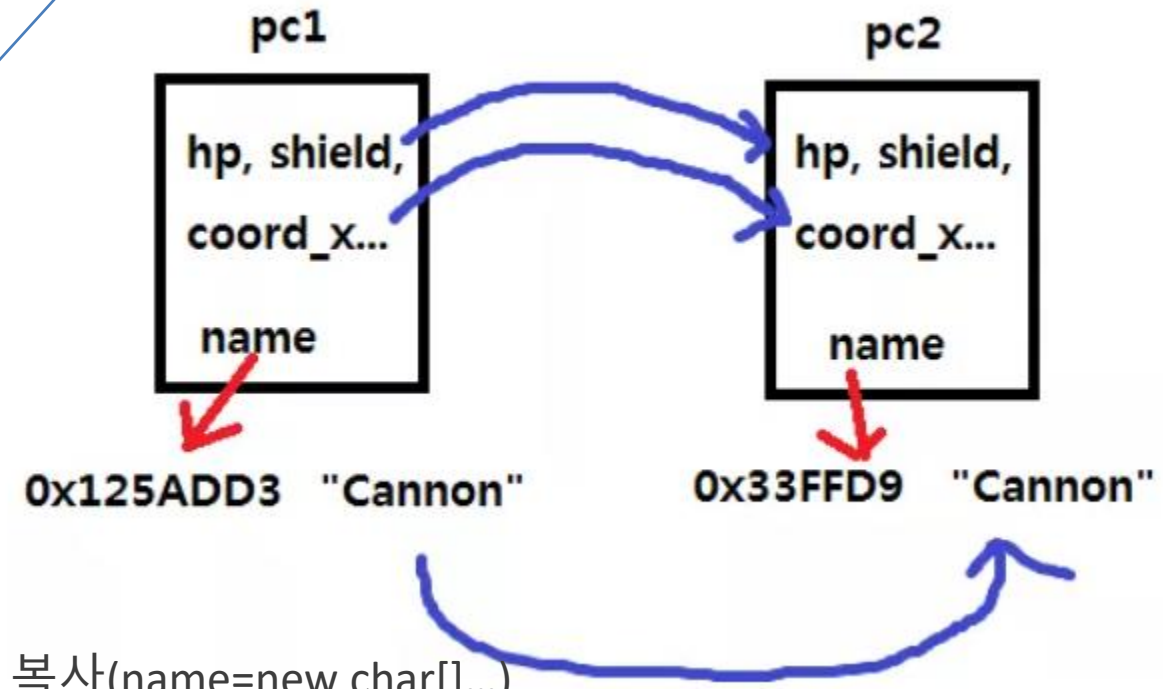


Main함수가 종료되고 Pc1에서 0x125ADD3 에 할당한 메모리 까지 delete하였으나 pc2에서 다시 한번 해제하여 런타임 오류

Default 생성자의 한계

복사 생성자에서 name 을 그대로 복사하지 말고 따로 다른 메모리에 동적 할당을 해서 그 내용만 복사

```
Photon_Cannon::Photon_Cannon(const Photon_Cannon &pc) {
  std::cout << "복사 생성자 호출! " << std::endl;
  hp = pc.hp;
  shield = pc.shield;
  coord_x = pc.coord_x;
  coord_y = pc.coord_y;
  damage = pc.damage;

  name = new char[strlen(pc.name) + 1];
  strcpy(name, pc.name);
}
```



깊은 복사(deep copy) : 메모리를 새로 할당해서 내용을 복사(name=new char[]…)

얕은 복사(shallow copy) :단순히 대입만 해주는 복사(hp, shield, coord_x,…)

컴파일러가 생성하는 디폴트 복사 생성자의 경우 얕은 복사 밖에 할 수 없으므로 위와 같이 깊은 복사가 필요한 경우에는 사용자가 직접 복사 생성자를 생성해야합니다.

Shallow copy vs Deep copy

- 얕은 복사 Shallow copy
  - 객체가 가진 멤버들의 값을 새로운 객체로 복사할 때,
    참조타입의 멤버를 가지고 있는 경우 참조값만 복사

- 깊은 복사 Deep copy
  - 전체 복사
  - 객체가 참조 타입의 멤버를 포함할 경우,
    참조값이 복사되는 것이 아닌 참조된 객체 자체가 새롭게 복사된다

Macro

- Macro 는 symbolic name을 갖는 코드의 조각
- #define 을 사용하여 symbolic한 이름을 부여
- #undef 를 사용하여 정의한 이름 해제
- Macro에 주어진 이름이 코드 내부에서 사용되는 경우, macro의 내용으로 대체된다
- 두 종류의 macro
  - Object-like macro
  - Function-like macro

Object-Like Macro

```cpp
#include <iostream>

#define PI 3.14

int main() {
    double radius = 10;
    double circumference = 2 * PI * radius;
    std::cout << circumference << std::endl;
}
```

```
62.8
```

Fuction-like Macro

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

#define SUB(x,y) x-y
#define ADD(x,y) x+y
#define PRINT(x) cout << x << endl;

int main() {
    int k = 10;
    int m = 5;
    int diff = SUB(k,m);
    int sum = ADD(k,m);
    PRINT(diff);
    PRINT(sum);
}
```

```
5
15
```

Macro vs Normal Function

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

#define ADD(x,y) x+y

int add(int x, int y) {
    return x + y;
}

int main() {
    int a = 30, b = 15, c = 3;
    cout << add(a,b)/c << endl;
    cout << ADD(a,b)/c << endl;
}
```

```
15
35
```

- Macro 는 컴파일되기 전에, 전처리기에서 Macro의 이름에 해당하는 부분을 Macro의 내용으로 바꾼다
- 코드의 짧은 조각에 이름을 부여하고 재사용하기 위해 사용

```cpp
cout << (a+b)/c << endl;
cout << a+b/c << endl;
```

Macro vs Normal Function

```c
#include <stdio.h>

#define SQR(X) X*X
#define PRT(X) printf("result : %d\n", X)

int main(void)
{
    int result;
    int x = 5;
    result = SQR(x);
    PRT(result);
    result = SQR(x+3);
    PRT(result);
}
```

```
x+3*x+3
```

```
result : 25
result : 23
```

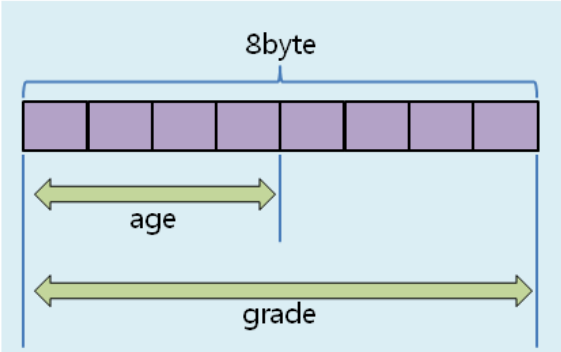
```c
#include <stdio.h>

#define SQR(X) ((X)*(X))
#define PRT(X) printf("result : %d\n", X)

int main(void)
{
    int result;
    int x = 5;
    result = SQR(x);
    PRT(result);
    result = SQR(x+3);
    PRT(result);
}
```

```
result : 25
result : 64
```

Union

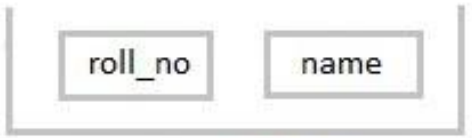
- Union은 class 혹은 struct와 비슷하지만,
  non-static data member를 한번에 하나만 가지고 있을 수 있다
  - 각각의 멤버는 그 클래스의 유일한 멤버인 것처럼 할당된다
  - Union의 크기는 가장 큰 데이터 멤버의 크기와 같다

- 메모리를 절약하기 위해 같은 데이터를 원하는 타입으로 선택하여 읽어오기 위해 사용



Structure

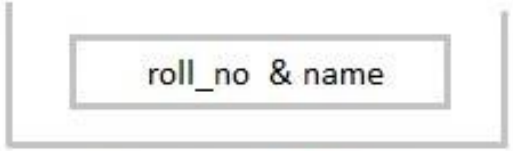
```
struct Student
{
  int roll_no;        // size 4 byte
  char name[40]; // size 40 byte
} s;
```



size of **s** will 44 bytes

Union

```
union Student
{
  int roll_no;        // size 4 byte
  char name[40]; // size 40 byte
} s;
```



size of **s** will 40 bytes

Union

```cpp
#include <iostream>

union test {
  int x;
  char c;
};

int main() {
  union test t;
  t.x = 65;
  std::cout << "size of t = " << sizeof(t) << std::endl;
  std::cout << "x = "<< t.x << std::endl;
  std::cout << "c = "<< t.c << std::endl;
}
```

```
size of t = 4
x = 65
c = A
```

ASCII 코드에 따라 65를 A로 출력

Union 사용 예제

IP 주소

- 32비트의 부호없는 정수
- 표기할 때에는 한 바이트씩 10진수로

- 0x46 = 70
- 0x0C = 12
- 0xDC = 220
- 0x21 = 33

-> 0x460CDC21 = 70.12.220.33

```
size of ip : 4
ip address in hex : 460cdc21
ip address : 70.12.220.33
```

```c
#include <stdio.h>

union IpAddr{
  unsigned int addr;
  struct{
    unsigned char ip4;
    unsigned char ip3;
    unsigned char ip2;
    unsigned char ip1;
  } ip_decimal;
};

int main() {
  IpAddr ip;
  ip.addr = 0x460CDC21; // == 70.12.220.33:
  printf("size of ip : %lu\n", sizeof(ip));
  printf("ip address in hex : %x\n",ip.addr);
  printf("ip address : %d.%d.%d.%d\n",
    ip.ip_decimal.ip1,
    ip.ip_decimal.ip2,
    ip.ip_decimal.ip3,
    ip.ip_decimal.ip4
  );
}
```

실습:최대공약수와 최소공배수

- 두 개의 자연수를 입력받아 최대 공약수와 최소 공배수를 출력하는 프로그램을 작성하시오
  - Macro에 함수를 사용하여 전처리

- 입력: 첫째중에는 두개의 자연수가 주어지며 자연수이며 사이에 한칸의 공백이 주어진다

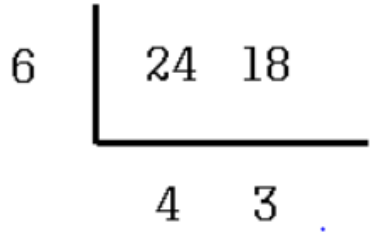
```
PS C:\vscodepractice\test_C++\helloworldcpp\cin\fishbreadp> ./main
12
16
```

- 출력: 첫째줄에는 입력으로 주어진 두수의 최대 공약수를, 둘째 줄에는 입력으로 주어진 두 수의 최소 공배수를 출력한다

```
최대공약수4
최소공배수48
```

실습:최대공약수와 최소공배수

- 최대공약수와 최소공배수 구하기



- 24와 18의 최대공약수: 6
- 24와 18의 최소공배수:6*4*3=72

```
#define min(a,b) (((a)<(b)) ? ((a) : (b)))
#include <iostream>
using namespace std;

int gcd(int x, int y) {
    int r;
        //macro min(a,b)를 사용하여 최대공약수를 구하는 식을 작성하세요.
    return r;
}

int main() {
    int a,b;
    cin >> a >> b;

    int gc = gcd(a,b);  // 최대공약수 gc

    int lc = ;  // 최소공배수를 구하는 식을 작성하세요.함수를 작성하셔도 됩니다.
    cout <<"최대공약수"<< gc << '\n' <<"최소공배수"<< lc;
    return 0;
}
```