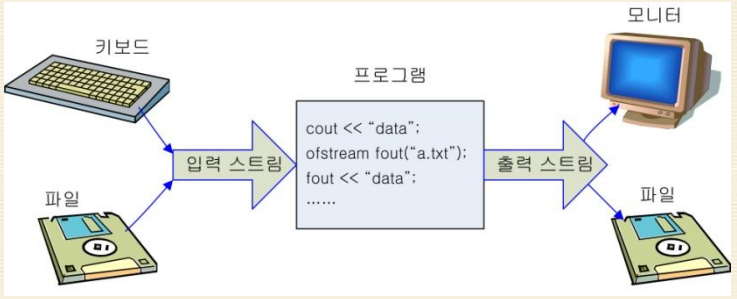
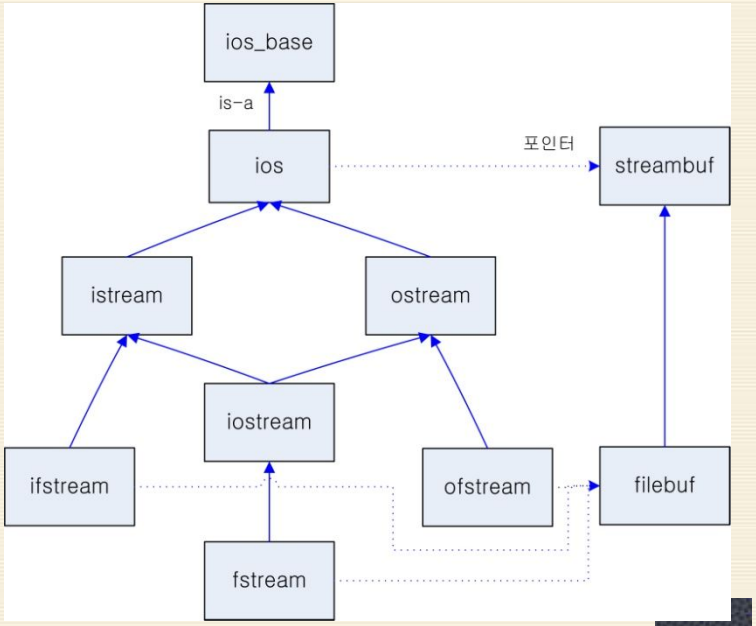
12장 파일 입출력

- 파일 입출력의 기초
- 파일 열기, 사용하기, 닫기
- 파일 입출력 모드
- 문자 단위 파일 입출력
- 텍스트 파일과 이진 파일
- read, write 함수에 의한 이진 파일 입출력
- 임의 접근
- 입출력 스트림 상태
- 입출력 연산자 오버로딩과 파일 입출력

1. 파일 입출력의 기초

- 파일 입출력 방법
  - 표준 입출력 객체인 cin, cout에 해당하는 파일 입출력 스트림 객체만 만들 수 있다면 기본적인 사용 방법은 표준 입출력과 동일
- 파일 입출력 스트림 객체의 생성
  - <fstream> 헤더 파일 필요
  - 입력 객체 : ifstream fin("in.dat");
  - 출력 객체 : ofstream fout("out.dat");
  - 입출력 객체 : fstream fio("data.dat");





2. 파일 열기, 사용하기, 닫기

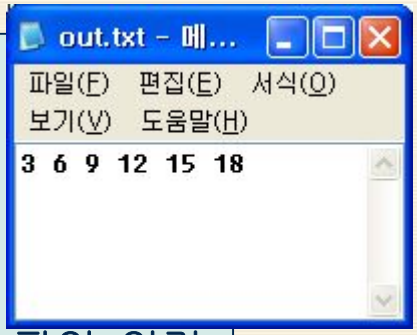
파일 출력 과정

1. 출력 스트림을 위한 ofstream 객체 생성
   - ofstream fout;
2. 생성된 ofstream 객체와 데이터를 출력할 파일 연결
   - fout.open(“out.dat”);
     - ✓ void open(const char *fName, int nMode);
     - ✓ fout.open("C:₩data₩out.dat", ios_base::out | ios_base::trunc);
       - nMode에 대해서는 다음 절에서 설명
   - 또는 1, 2를 동시에 수행 : ofstream fout(“out.dat”);
3. 표준 출력 객체(cout)와 동일한 방법으로 사용
   - fout << “test” << endl;
4. 파일 연결 해제
   - fout.close();
     - ✓ fout 객체 자체가 소멸되는 것은 아님
       - → fout.open()을 통해 또 다른 파일 연결 가능

2. 파일 열기, 사용하기, 닫기

- 출력 예 : 1부터 20까지의 정수 중 3의 배수를 “out.txt”로 출력

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

int main(void)
{
    ofstream fout("out.txt");        // 출력 스트림 생성 및 파일 열기

    if (!fout) {                     // 에러 처리
        cout << "파일 열기 에러" << endl;
        return 1;
    }

    for (int i = 1; i <= 20; i++) {
        if (i % 3 == 0)
            fout << i << " ";        // 표준 출력과 동일하게 사용
    }

    fout.close();

    return 0;
}
```



파일 열기 도중 에러 발생
→ **failbit**이 **1**로 변경됨 **(11.9절)**
ofstream 클래스는 **!** 연산자 오버로딩을 통해 **fail**인 경우 **true**를 반환

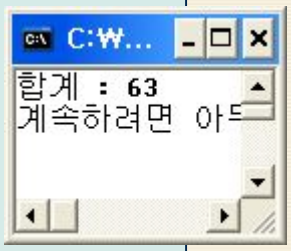
if (fout == NULL) { … } 와 같이 사용 가능
fout은 변환함수**(15.2절)**를 통해 **fail**인 경우 **NULL**로 변환될 수 있음

2. 파일 열기, 사용하기, 닫기

- 입력 예 : 파일 끝(EOF)을 만날 때까지 값을 읽어 합계를 출력

```
int main(void)
{
    int Num;
    int Sum = 0;
    ifstream fin("out.txt");          // 입력 스트림 생성 및 파일 열기

    if (fin == NULL) {
        cout << "파일 열기 에러" << endl;
        return 1;
    }

    fin >> Num;

    while (!fin.eof()) {
        Sum += Num;
        fin >> Num;
    }

    cout << "합계 : " << Sum << endl;

    fin.close();

    return 0;
}
```

cin의 사용 방법과 동일



3. 파일 입출력 모드

- void open(const char *fName, int nMode);
  - 파일 입출력 모드
    - 파일 열기 모드 : 읽기, 쓰기 등
    - 접근 모드 : 텍스트 모드, 이진 모드

상수	의미
ios_base::in	입력을 위한 파일 열기
ios_base::out	출력을 위한 파일 열기
ios_base::ate	파일을 열 때 파일의 끝 위치를 찾아 끝 위치로 이동 파일의 입출력은 파일의 모든 위치에서 가능
ios_base::app	파일에 대한 출력은 파일의 끝에 추가 파일의 출력은 파일의 마지막 위치에만 가능하지만 입력은 모든 위치에서 가능
ios_base::trunc	동일한 이름의 파일이 존재하면 파일의 모든 내용을 삭제
ios_base::binary	이진 형식으로 파일 열기

  - 파일 입출력 관련 클래스들의 디폴트 입출력 모드

클래스	디폴트 파일 열기 모드
ifstream	ios_base::in
ofstream	ios_base::out | ios_base::trunc
fstream	ios_base::in | ios_base::out

3. 파일 입출력 모드

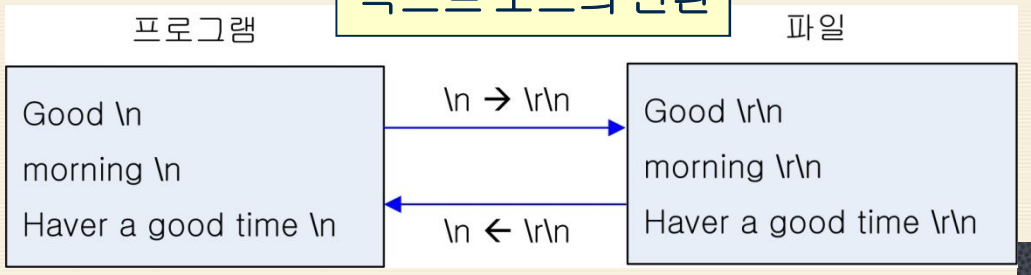
- C와 C++의 파일 입출력 모드 비교

out만 설정하면 자동으로 **trunc** 처리가 됨

C++ 모드					C 모드
binary	in	out	trunc	app	
		●			"w"
		●		●	"a"
		●	●		"w"
	●				"r"
	●	●			"r+"
	●	●	●		"w+"
●		●			"wb"
●		●		●	"ab"
●		●	●		"wb"
●	●				"rb"
●	●	●			"r+b"
●	●	●	●		"w+b"

- 텍스트 모드와 이진 모드의 차이
  - 줄바꿈 문자의 변환
    - 캐리지 리턴(₩n)
    - 라인 피드(₩r)

텍스트 모드의 변환

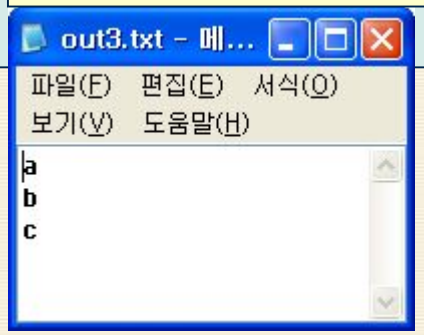


3. 파일 입출력 모드

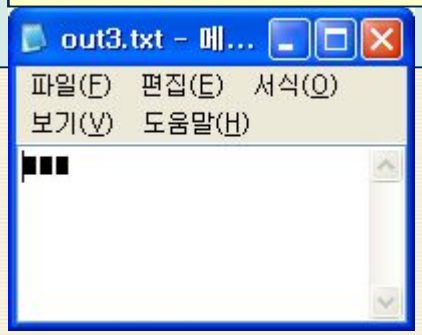
⌗ 예 : 텍스트 모드와 이진 모드의 차이

```
int main(void)
{
    ofstream fout("out3.txt", ios_base::out | ios_base::trunc | ios_base::binary);
    fout << 'a' << endl << 'b' << endl << 'c' << endl;

    fout.close();

    return 0;
}
```

텍스트 모드
ios_base::binary를 삭제한 경우



이진 모드
ios_base::binary를 추가한 경우



일반적인 데이터 처리 방법
- 텍스트 모드로 저장 → 텍스트 모드로 읽기
- 이진 모드로 저장 → 이진 모드로 읽기

4. 문자 단위 파일 입출력

문자 단위 입출력 함수들

- 기본적인 사용 방법은 11.7절, 11.8절과 동일

멤버 함수	설명
int get(void);	문자 1개 입력
istream &get(char &)	문자 1개 입력, 연속 호출 가능
ostream &put(char)	문자 1개 입력, 연속 호출 가능
istream &get(char *, int, char = '₩n')	지정한 개수 또는 줄 단위 문자열 입력, '₩n' 제거하지 않음
istream &getline(char *, int, char = '₩n');	지정한 개수 또는 줄 단위 문자열 입력, '₩n' 제거

기타 문자 입력 관련 멤버 함수

- int peek(void); // 단지 다음 문자가 무엇인지 읽어옴 (제거 X)
- istream &putback(char c); // 마지막 입력 문자(c)를 되돌림

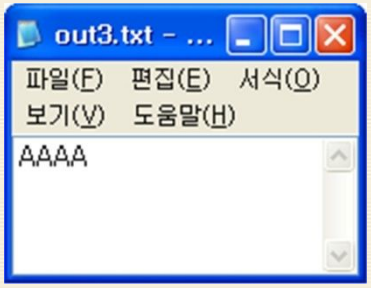
4. 문자 단위 파일 입출력

- 예 : get, put 함수를 이용한 파일 복사

```
int main(void)
{
    ifstream fin("in4.txt");
    ofstream fout("out4.txt");

    if (!fin || !fout) {
        cout << "파일 열기 에러" << endl;
        return 1;
    }

    char ch = fin.get();          // 문자 하나 입력

    while (!fin.eof()) {
        fout.put(ch);             // 문자 하나 출력
        fin.get(ch);
    }

    fin.close();
    fout.close();

    return 0;
}
```

5. 텍스트 파일과 이진 파일

- 일상적인 컴퓨터 사용에서의 텍스트 파일과 이진 파일
  - 텍스트 파일 : 메모장에서 작성한 파일
  - 이진 파일 : MS 워드 파일, 아래아 한글 파일 등
- 프로그래밍 관점에서의 텍스트 파일과 이진 파일
  - 텍스트 파일 : 데이터 저장 시 모든 데이터를 문자로 변환하여 저장하는 것
    - 예 : int a = 15; → '1'과 '5'라는 문자로 저장 → 2바이트
  - 이진 파일 : 데이터 저장 시 메모리의 형태 그대로 저장하는 것
    - 예 : int a = 15; → 메모리 형태 그대로 저장 → 4바이트
- 다음 파일은 텍스트 파일인가 이진 파일인가?
  - 이 파일을 만든 프로그래머만이 대답 가능
    - 텍스트 파일 : 'A', 'A', 'A', 'A' 4개의 문자 출력
    - 이진 파일 : int a = 1094795585; 를 4바이트 형태 그대로 출력
      - ✓ 0X41414141 → AAAA로 출력됨



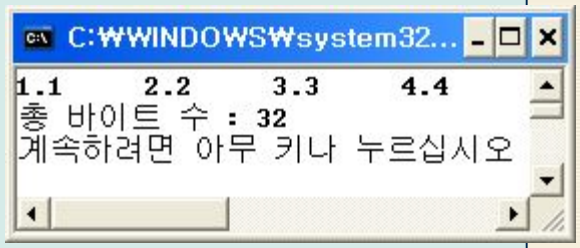
5. 텍스트 파일과 이진 파일

- 텍스트 파일과 이진 파일을 만드는 방법
  - 파일 스트림 클래스의 멤버 함수에 따라 달라짐
  - 텍스트 파일로 저장 : << 연산자
  - 텍스트 파일로부터 입력 : >> 연산자
  - 이진 파일로 저장 : write
  - 이진 파일로부터 입력 : read
- 주의 사항
  - 파일 열기 모드(텍스트 모드, 이진 모드)와 텍스트 파일 및 이진 파일은 원칙적으로 무관함
  - 그러나, 일반적으로 텍스트 파일은 텍스트 모드로 열고, 이진 파일은 이진 모드로 열어 작업 진행

6. read, write 함수에 의한 이진 파일 입출력

- read, write 함수 프로토타입
  - istream &read(unsigned char *buffer, int size);
  - ostream &write(const unsigned char *buffer, int size);
    - buffer : 출력 또는 입력 데이터를 저장할 변수의 주소
    - size : 데이터의 크기(바이트 단위)

```
int main(void)
{
    ofstream fout("ex5.dat", ios_base::out | ios_base::binary); // 출력 스트림
    double nums[4] = { 1.1, 2.2, 3.3, 4.4 };
    fout.write((char *) nums, sizeof(nums));  // double(8)*4개=32바이트 출력
    fout.close();

    ifstream fin("ex5.dat", ios_base::in | ios_base::binary);
    fin.read((char *) nums, sizeof(nums));
    for (int i = 0; i < 4; i++)
        cout << nums[i] << '₩t';
    cout << endl;

    cout << "총 바이트 수 : " << fin.gcount() << endl;

    return 0;
}
```



6. read, write 함수에 의한 이진 파일 입출력

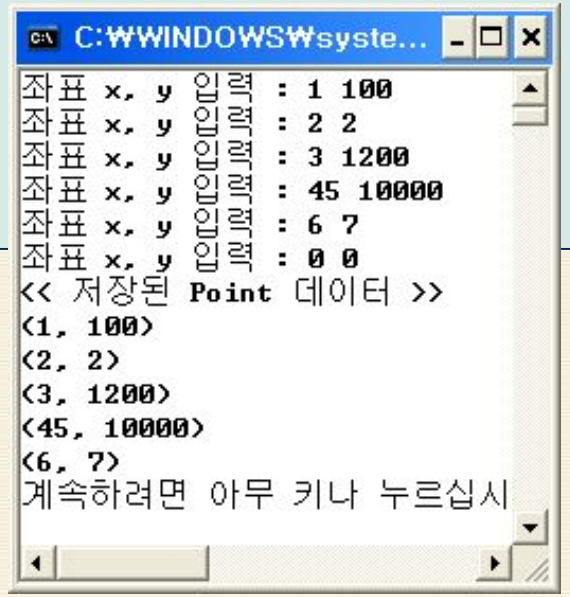
구조체 변수의 이진 파일 입출력 예

```
struct Point {
    int x, y;
};

int main(void)
{
    Point Po;

    ofstream fout("ex6.dat",ios_base::out | ios_base::app | ios_base::binary);

    cout << "좌표 x, y 입력 : ";
    cin >> Po.x >> Po.y;

    while (Po.x != 0 && Po.y != 0) {
        fout.write((char *) &Po, sizeof(Point));        // Point 크기(8)만큼 출력
        cout << "좌표 x, y 입력 : ";
        cin >> Po.x >> Po.y;
    }

    fout.close();
```

6. read, write 함수에 의한 이진 파일 입출력

⌗ 구조체 변수의 이진 파일 입출력 예 (계속)

```
    cout << "<< 저장된 Point 데이터 >>" << endl;

    ifstream fin("ex6.dat", ios_base::in | ios_base::binary);

    while (fin.read((char *) &Po, sizeof(Point))) {
        cout << "(" << Po.x << ", " << Po.y << ")" << endl;
    }

    fin.close();

    return 0;
}
```



7. 임의 접근

- 스트림 포인터
  - 입력 포인터(get pointer) : 다음 입력이 수행되는 위치를 가리킴
  - 출력 포인터(put pointer) : 다음 출력이 수행될 위치를 가리킴
  - 입력과 출력이 진행되면 자동으로 입출력된 바이트 수만큼 다음 위치로 이동함
- 입력 포인터와 출력 포인터를 임의의 위치로 이동하는 방법
  - seekg, seekp 함수 사용

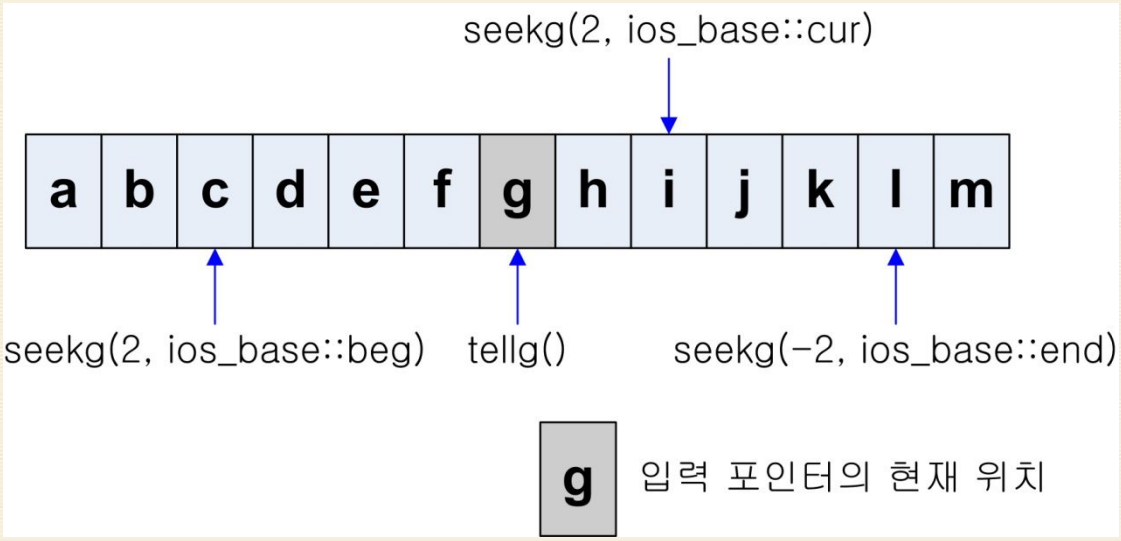
```
istream &seekg(streampos pos);
ostream &seekp(streampos pos);
istream &seekg(streamoff off, ios_base::seek_dir dir);
ostream &seekp(streamoff off, ios_base::seek_dir dir);
```

streampos, streamoff는 long 타입과 동일

값	의미
ios_base::beg	파일의 처음 위치를 기준으로 새로운 위치로 이동
ios_base::cur	파일의 현재 위치를 기준으로 새로운 위치로 이동
ios_base::end	파일의 마지막 위치를 기준으로 새로운 위치로 이동

7. 임의 접근

- 입력 포인터와 출력 포인터의 현재 위치를 확인하는 방법
  - streampos tellg();
  - streampos tellp();
- seekg와 tellg 함수의 적용 예

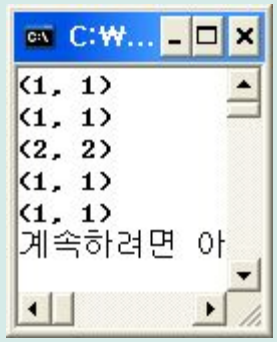


- 임의 접근은 이진 파일에 보다 적합함
  - int 값들이 저장되어 있는 경우 50번째 int 값으로 이동
  - fin.seekg((50 - 1) * 4); // 이진 파일인 경우 가능

7. 임의 접근

- 예 : (1, 1)의 좌표를 5개 저장한 후 세 번째 좌표를 (2, 2)로 변경

```
struct Point {
    int x, y;
};

int main(void)
{
    Point Po = { 1, 1 };

    fstream fio("ex7.dat", ios_base::in | ios_base::out | ios_base::trunc |
                                ios_base::binary);
    for (int i = 0; i < 5; i++)
        fio.write((char *) &Po, sizeof(Point));

    fio.seekp(2 * sizeof(Point), ios_base::beg);        // 출력 포인터를 3번째 원소로
    Po.x = 2; Po.y = 2;
    fio.write((char *) &Po, sizeof(Point));

    fio.seekg(0, ios_base::beg);     // 입력 포인터를 첫 번째 원소로 이동
    while (fio.read((char *) &Po, sizeof(Point))) {
        cout << "(" << Po.x << ", " << Po.y << ")" << endl;
    }
    fio.close();

    return 0;
}
```



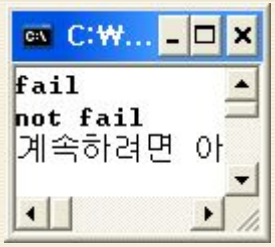
8. 입출력 스트림 상태

파일 입출력 스트림 상태

- 표준 입출력 스트림 상태의 처리 방법과 동일 : 11.9절

```
int main(void) {
    int a;

    ifstream fin("ex8.dat");
    fin >> a;

    if (fin.fail())
        cout << "fail" << endl;
    else
        cout << "not fail" << endl;

    fin.clear();

    if (fin.fail())
        cout << "fail" << endl;
    else
        cout << "not fail" << endl;

    fin.close();

    return 0;
}
```

의도하는 타입의 데이터를 읽지 못하는 경우 → **fail**



9. 입출력 연산자 오버로딩과 파일 입출력

- CPoint 클래스에 대한 입출력 연산자 오버로딩 : 7.9절
  - CPoint Po;
  - cin >> Po;
  - cout << Po;
- 다음과 같은 방식의 파일 입출력을 원한다면?
  - fin >> Po;
  - fout << Po;
- 원리

별도로 해야 할 작업은 없음
→ **istream, ostream** 클래스에 대한
입출력 연산자 오버로딩만으로 충분

```
ostream &operator<<(ostream &os, CPoint &Po)
{
    os << "(" << Po.x << ", " << Po.y << ")" << endl;
    return os;
}
```

ostream 클래스는 **ofstream** 클래스의 **base** 클래스이므로
fout << Po;에서 **fout** 객체를 받을 수 있음.
결국 **ofstream** 객체인 **fout**으로 출력됨.

9. 입출력 연산자 오버로딩과 파일 입출력

- CPoint 클래스에 대한 입출력 연산자 오버로딩 예

```
class CPoint {
private :
    int x, y;

public :
    CPoint(int a = 0, int b = 0) : x(a), y(b) { }
    friend istream &operator>>(istream &is, CPoint &Po);
    friend ostream &operator<<(ostream &os, CPoint &Po);
};

istream &operator>>(istream &is, CPoint &Po)  //  >> 연산자 오버로딩
{
    is >> Po.x >> Po.y;
    return is;
}

ostream &operator<<(ostream &os, CPoint &Po)  //  << 연산자 오버로딩
{
    os << Po.x << " " << Po.y << endl;
    return os;
}
```

9. 입출력 연산자 오버로딩과 파일 입출력

CPoint 클래스에 대한 입출력 연산자 오버로딩 예 (계속)

```
int main(void) {
    int i;
    CPoint Po[5];

    for (i = 0; i < 5; i++) {
        cout << "좌표(x, y) 입력 : ";
        cin >> Po[i];
    }

    ofstream fout("ex9.dat");
    for (i = 0; i < 5; i++)
        fout << Po[i];              //  << : ofstream 객체에도 적용 가능
    fout.close();

    ifstream fin("ex9.dat");
    CPoint Temp;
    for (i = 0; i < 5; i++) {
        fin >> Temp;
        cout << Temp;;
    }
    fin.close();

    return 0;
}
```