16강. 파일 입출력

C++ 프로그래밍

jhhwang@kumoh.ac.kr

컴퓨터공학과 황준하

목차

파일 입출력 기초

- 파일 입출력 과정
  - 파일 스트림 객체 생성
  - 파일 열기
  - 사용 : 기본적으로 표준 입출력 객체(cin, cout) 사용 방법과 동일
  - 파일 닫기
- 파일 스트림 클래스의 종류
  - 파일 출력 스트림 클래스 : ofstream
  - 파일 입력 스트림 클래스 : ifstream
  - 파일 입출력 스트림 클래스 : fstream

파일 입출력 기초

- 파일 출력 예

파일 스트림 클래스 포함

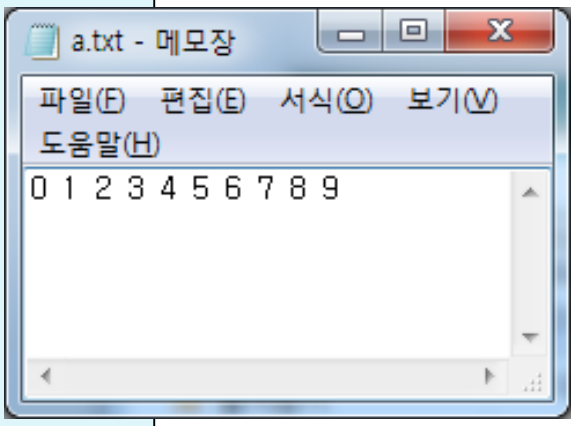
파일 출력 객체 생성

파일 열기(연결)

cout과 동일하게 사용

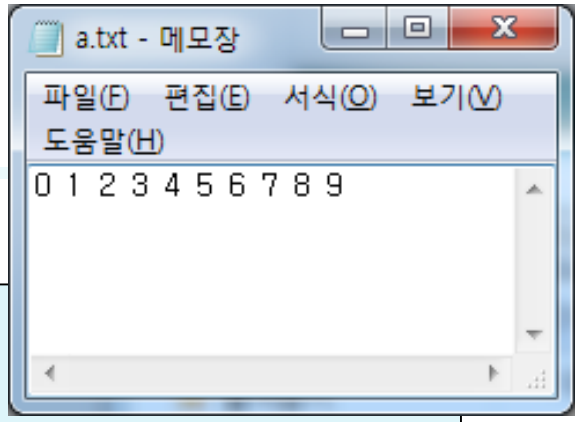
파일 닫기(연결 해제)

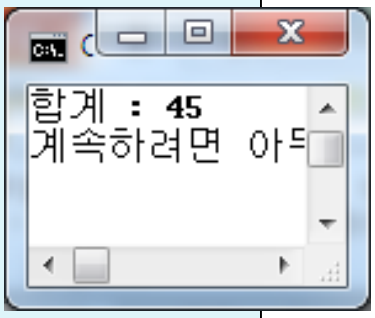
```
#include <iostream>
#include <fstream>
using namespace std;

void main(void)
{
    ofstream fout;

    fout.open("a.txt");

    for (int i = 0; i < 10; i++)
        fout << i << " ";
    fout << endl;

    fout.close();
}
```



파일 입출력 기초

- 파일 입력 예
  - a.txt로부터 정수 읽기





파일 입력 객체 생성

파일 열기(연결)

cin과 동일하게 사용

파일 닫기(연결 해제)

```
void main(void)
{
    ifstream fin;

    fin.open("a.txt");

    int Sum = 0;
    int Num;
    for (int i = 0; i < 10; i++) {
        fin >> Num;
        Sum += Num;
    }

    cout << "합계 : " << Sum << endl;

    fin.close();
}
```

파일 입출력 모드

- 입출력 스트림 클래스의 생성자
  - 스트림 객체 생성과 동시에 파일 열기(연결) 가능
  - ofstream(char *filename, int mode = ios_base::out)
  - ifstream(char *filename, int mode = ios_base::in)
  - fstream(char *filename, int mode = ios_base::in | ios_base::out)
- 파일 입출력 모드
  - 파일 : 공유 자원
    - 파일 사용 권한 획득 필요
    - 읽기, 쓰기, 읽기쓰기 등 모든 설정

```
void main(void)
{
	ofstream fout("a.txt");

	fout << 123 << endl;

	fout.close();
}
```

파일 입출력 모드

- 파일 입출력 모드
  - fio.open("filename", Mode);
  - 파일 열기 모드 : 읽기, 쓰기 등
  - 파일 접근 모드 : 텍스트, 이진

모드(상수)	의미
ios_base::in	입력
ios_base::out	출력, 이 모드만 설정 시 자동으로 ios_base::trunc 적용
ios_base::ate	파일 open 시 입출력 포인터(현재 읽기쓰기 위치)를 파일 끝으로 이동, 모든 위치에서 읽고 쓰고 가능
ios_base::app	파일 읽기는 모든 위치에서 가능, 쓰기는 파일의 마지막 위치에 추가만 가능
ios_base::trunc	해당 파일이 존재할 경우 파일의 내용 모두 삭제
ios_base::binary	이진 모드로 열기

파일 입출력 모드

Ios_base::app 사용 예

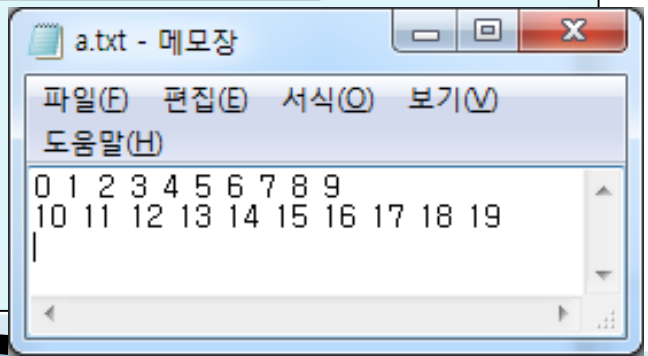
```
void main(void)
{
	fstream fio("a.txt", ios_base::in | ios_base::out | ios_base::app);

	for (int i = 10; i < 20; i++)
		fio << i << " ";
	fio << endl;

	fio.seekg(0, ios_base::beg);

	int Sum = 0;
	int Num;
	for (int i = 0; i < 20; i++) {
		fio >> Num;
		Sum += Num;
	}
	cout << "합계 : " << Sum << endl;
	fio.close();
}
```

마지막에 추가

데이터 추가

입력 포인터(읽기 위치)를 처음으로 이
→ 임의 접근에서 자세히 설명

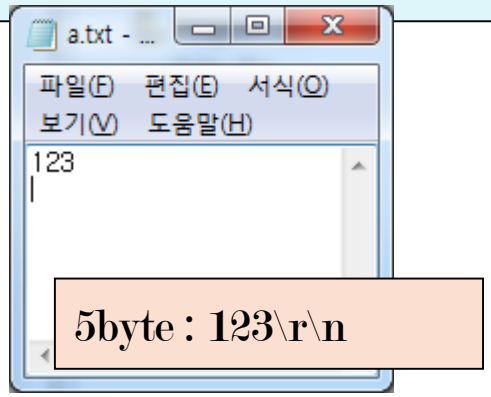
데이터 읽기 및 확인



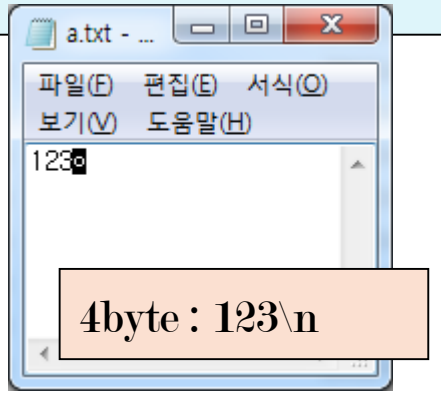
파일 입출력 모드

- 텍스트 모드와 이진 모드
  - 텍스트 모드 : \n → \r\n 두 자로 변환하여 처리
  - 이진 모드 : \n ←→ \n 그대로 처리

```
void main(void)
{
	ofstream fout("a.txt");
	fout << "123" << endl;
	fout.close();
}
```

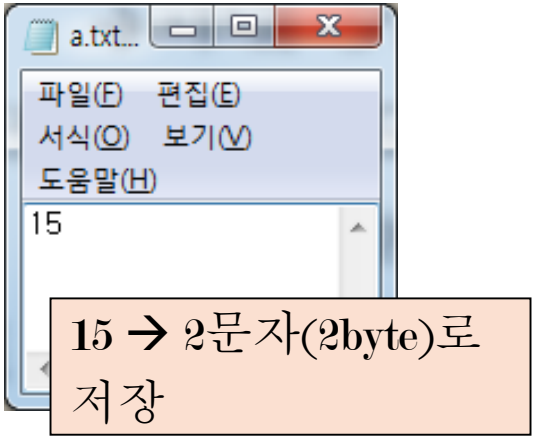



```
void main(void)
{
	ofstream fout("a.txt",
	  ios_base::out | ios_base::binary);
	fout << "123" << endl;
	fout.close();
}
```

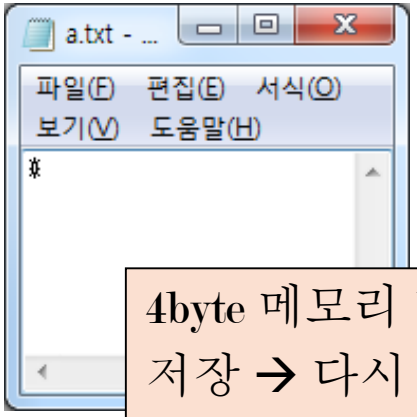


텍스트 파일과 이진 파일

- 메모리의 데이터를 파일로 저장하는 방법
  - 문자화하여 사람이 읽기 쉽게 저장 → 텍스트 파일
  - 메모리의 내용 그대로 파일로 저장 → 이진 파일
- 예 : int a = 15;에서 a의 값 15를 파일로 저장하는 방법



15 → 2문자(2byte)로 저장



4byte 메모리 형태 그대로 저장 → 다시 4byte만큼 읽어서 int 변수에 저장하면 15임

텍스트 파일과 이진 파일

- 텍스트 파일과 이진 파일을 만들고 사용하는 방법
  - 파일 스트림 클래스의 멤버 함수의 차이
  - << 연산자 : 텍스트 파일로 출력
  - >> 연산자 : 텍스트 파일로 입력
  - write 함수 : 이진 파일로 저장
  - read 함수 : 이진 파일로 입력
- 참고 사항
  - 파일 열기 모드(텍스트 모드, 이진 모드)와 텍스트 파일, 이진 파일은 원칙적으로 무관함
  - 그러나, 일반적으로 텍스트 파일은 텍스트 모드로 열고, 이진 파일은 이진 모드로 열어 작업

이진 파일 입출력

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

```
void main(void)
{
	ofstream fout("a.txt", ios_base::out | ios_base::binary);

	int a = 15;
	fout.write((char *) &a, sizeof(int));

	fout.close();
}
```

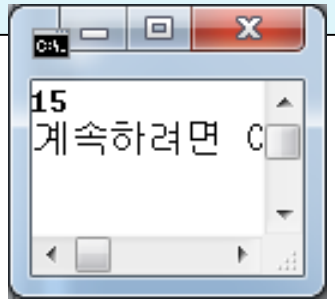
a의 시작주소로부터 시작해서 4바이트(int) 만큼을 파일로 저장 → a의 값을 메모리 형태 그대로 저장

이진 파일 입출력

- read 함수 사용 예

```
void main(void)
{
	ifstream fin("a.txt", ios_base::in | ios_base::binary);

	int a;
	fin.read((char *) &a, sizeof(int));
	cout << a << endl;

	fin.close();
}
```

4바이트(int)만큼 파일로부터 데이터를 읽어 메모리의 변수 a의 시작주소에 저장함
→ int 값 하나를 읽어 a에 저장

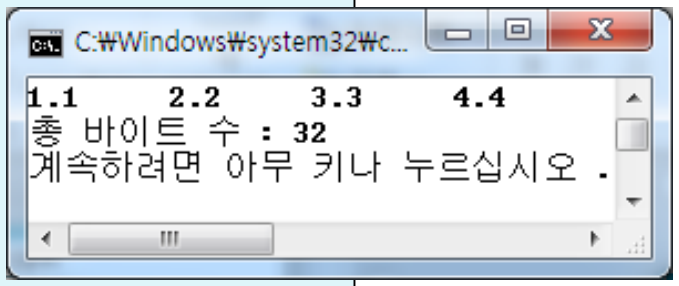


이진 파일 입출력

- 배열 데이터의 이진 파일 입출력 예

```
void main(void)
{
	ofstream fout("a.txt", ios_base::out | ios_base::binary);
	double nums[4] = { 1.1, 2.2, 3.3, 4.4 };
	fout.write((char *) nums, sizeof(nums));
	fout.close();

	ifstream fin("a.txt", ios_base::in | ios_base::binary);
	double nums2[4];
	fin.read((char *) nums2, sizeof(nums2));
	for (int i = 0; i < 4; i++)
		cout << nums2[i] << '\t';
	cout << endl;

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

배열의 크기(32바이트)만큼 쓰기



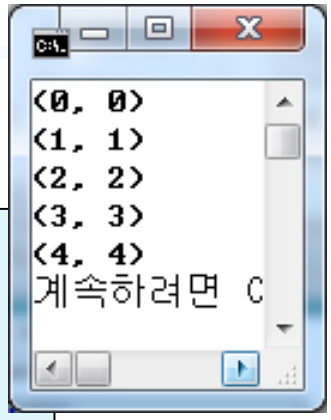
마지막으로 읽은 데이터의 바이트 수

이진 파일 입출력

- 구조체 변수의 이진 파일 입출력 예

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

```
void main(void)
{
        Point Po;

        ofstream fout("a.txt",ios_base::out | ios_base::binary);
        for (int i = 0; i < 5; i++) {
                Po.x = Po.y = i;
                fout.write((char *) &Po, sizeof(Point));
        }
        fout.close();

        ifstream fin("a.txt", ios_base::in | ios_base::binary);
        while (fin.read((char *) &Po, sizeof(Point))) {
                cout << "(" << Po.x << ", " << Po.y << ")" << endl;
        }
        fin.close();
}
```



Po 데이터(8바이트)를 파일로 저장

하나씩 다시 읽어오기 : 더 이상 읽을 데이터 없을 경우 while문을 빠져나오게 됨

임의 접근

- 스트림 포인터
  - 입력 포인터(get pointer) : 다음 입력 위치를 가리킴
  - 출력 포인터(put pointer) : 다음 출력 위치를 가리킴
  - 입력과 출력이 진행되면 자동으로 입출력된 바이트 수 만큼 다음 위치로 이동함
- 입력 포인터와 출력 포인터의 위치를 이동하는 방

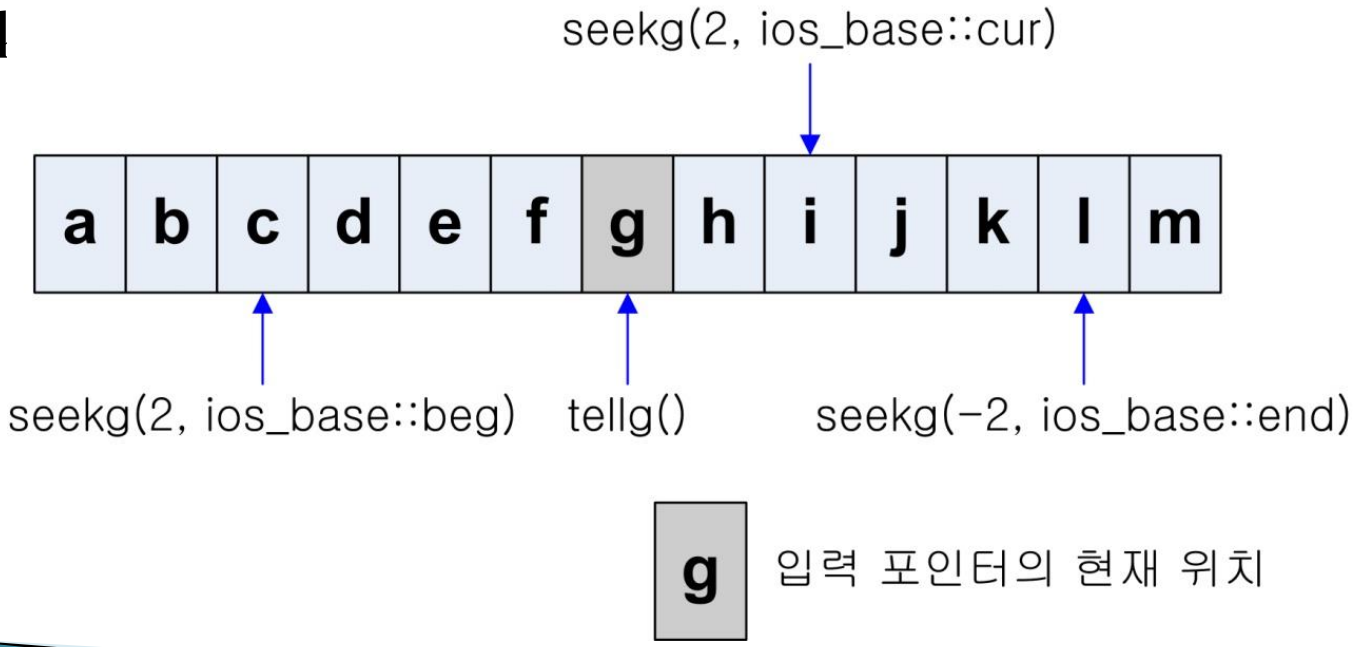
```
fin.seekg(위치_바이트단위);
fin.seekg(위치_바이트단위, 방향기준);
fout.seekp(위치_바이트단위);
fout.seekp(위치_바이트단위, 방향기준);
```

방향기준

ios_base::beg : 파일 시작 위치
ios_base::cur : 현재 위치
ios_base::end : 마지막 (다음) 위치

임의 접근

- 입력 포인터와 출력 포인터의 현재 위치를 확인하는 함수
  - fin.tellg()
  - fout.tellp()
- seel



임의 접근

- 임의 접근의 활용
  - 데이터의 크기가 일정한 이진 파일에 대한 활용도가 높음

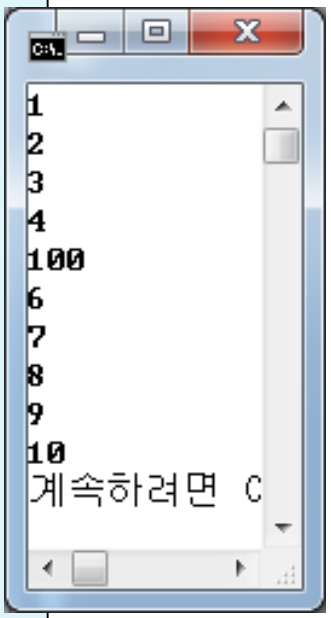
```
ofstream fout("a.txt", ios_base::out | ios_base::binary);
for (int i = 1; i <= 10; i++)
      fout.write((char *) &i, sizeof(int));

int Num = 100;
fout.seekp((5 - 1) * sizeof(int), ios_base::beg);
fout.write((char *) &Num, sizeof(int));

fout.close();

ifstream fin("a.txt", ios_base::in | ios_base::binary);
for (int i = 0; i < 10; i++) {
      fin.read((char *) &Num, sizeof(int));
      cout << Num << endl;
}
fin.close();
```

1~10 데이터 쓰기

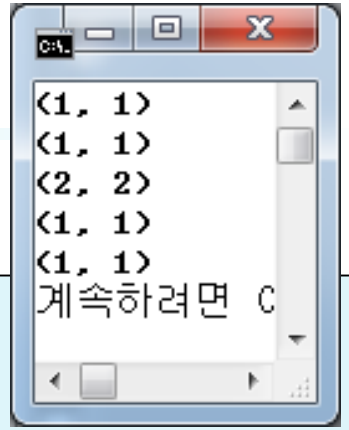
5번째 데이터로 이동

값 100 쓰기

값들을 읽고 확인



임의 접근



- 구조체 데이터에 대한 임의 접근 예

```
void main(void)
{
	Point Po = { 1, 1 };
	fstream fio("a.txt", 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((3 - 1) * sizeof(Point), ios_base::beg);
	Po.x = 2; Po.y = 2;
	fio.write((char *) &Po, sizeof(Point));

	fio.seekg(0, ios_base::beg);
	while (fio.read((char *) &Po, sizeof(Point)))
		cout << "(" << Po.x << ", " << Po.y << ")" << endl;

	fio.close();
}
```

(1, 1) 5개 저장

출력 포인터를 세번 째 데이터로 이동

(2, 2)로 변경

입력 포인터를 시작위치로 이동

저장된 데이터 확인