

Lab 5

File I/O and Vector

<https://padlet.com/dcslabcp/0412-szvffj0wkmoq161m>

Edit tasks.json (Windows)

- File I/O를 실습하기 전, tasks.json의 끝에
“options”: {“cwd”: “\${fileDirname}”} 를 추가

```
// 바이너리 실행(Windows)
{
  "label": "execute",
  "command": "cmd",
  "group": "test",
  "args": [
    "/C", "${fileDirname}\\${fileBasenameNoExtension}"
  ],
  "options": {"cwd": "${fileDirname}"}
}
```

C++ File I/O

- C++ file I/O are easy to implement using the class of ofstream and ifstream in the **fstream** library.

<File IO Classes>

- ofstream: File classes for **write operations**
- ifstream: File classes for **read operations**
- **fstream**: File classes for **both** reads and writes (derived from iostream)

File I/O options

ios::in	Open for input operations.	ifstream default parameter
ios::out	Open for output operations.	ofstream default parameter
ios::binary	Open in binary mode.	
ios::app (append)	All output operations are performed at the end of the file, appending the content to the current content of the file.	
ios::trunc (overwrite)	If the file is opened for output operations and it already existed, its previous content is deleted and replaced by the new one.	

사용법

- 2가지 방법

- 각 Class의 Constructor를 사용

e.g. ifstream myFile (“*원하는 File 경로*”);

- open() member function을 사용

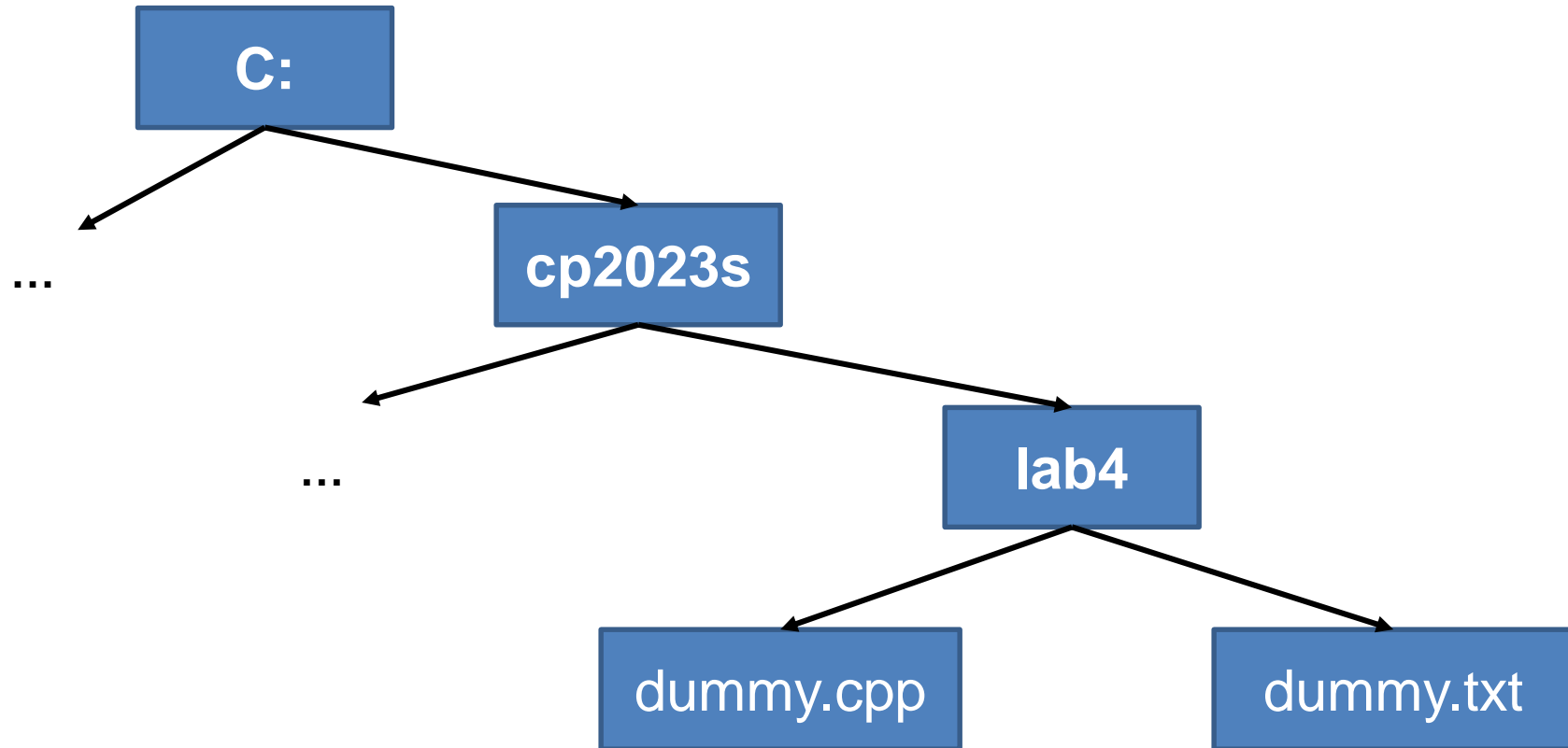
e.g. ifstream myFile; myFile.open (“*원하는 File 경로*”);

- 만약 file이 없었다면 알아서 Create

File 경로

- File에 접근하는 방법 == File name을 표시하는 법
- 절대 경로: File Tree의 맨 끝에서부터 표현
e.g. C:\\User\\cp2023s\\lab04\\dummy.txt
- 상대 경로: File에 접근하는 Source Code의 위치를 기준으로 생각
 - 같은 Folder에 있다면 바로 file 명을 작성
e.g. dummy.txt

File 경로

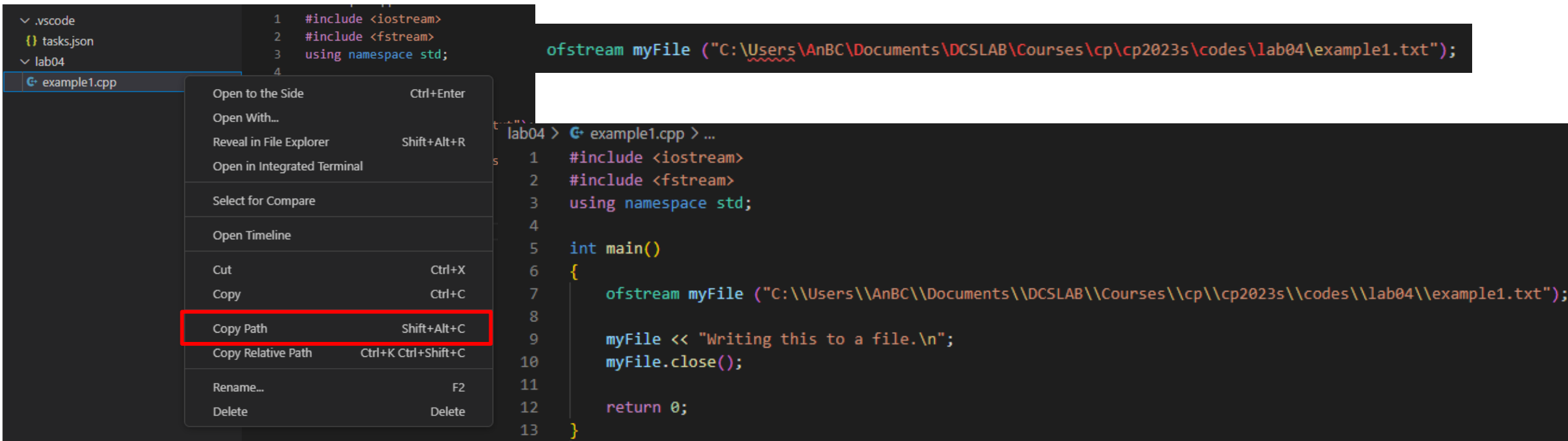


About Files

- EOF = End Of File ← File의 끝에 들어있는 표시자
 - eof를 이용해 File 끝에 도달했는지 확인
- `is_open(file class)` = file이 제대로 open 되었다면 `true`를 return
- `getline(file class, buffer)` = file을 한 줄 씩 읽어서 buffer에 할당
 - EOF에 도달하면 `false`를 return
 - 주로 `while()`과 함께 사용
- `close(file class)` = open한 file을 close

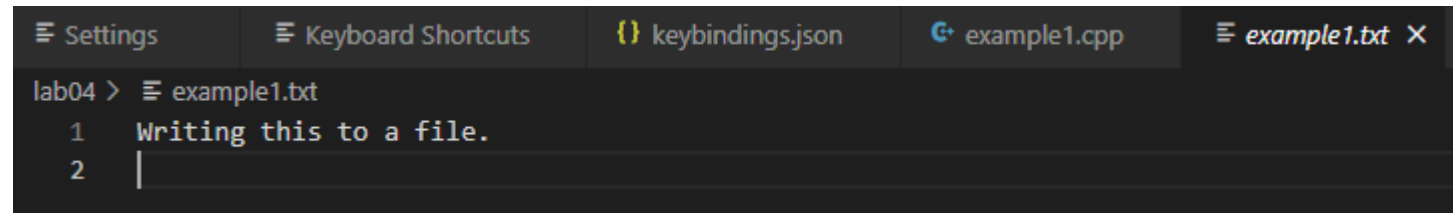
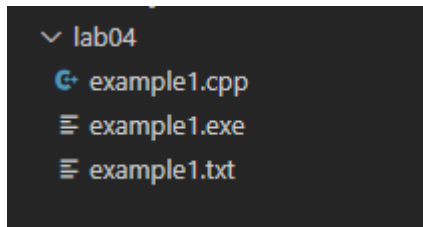
Example 1 – File Write

- ‘example1.cpp’ file을 생성 후 ofstream class를 이용해 ‘example1.txt’ file 생성 *Windows만
- 절대 경로를 이용 ← ‘example1.cpp’를 우클릭 후 ‘copy path’ 이용 + “\” 1개씩 더 추가



Example 1

- example1.txt가 생성된 것을 확인해보고 내용을 확인해보자



Example 2 – File Read

- ‘example2.cpp’ file을 생성 후 ifstream class를 이용해 ‘example1.txt’ file 을 읽자
- 상대 경로를 이용
 - string class를 이용하기 위해 <string>을 include
 - close()를 해주지 않아도 class destructor가 정리해 줌
 - 그래도 close() function을 이용해 닫는 습관이 좋음

lab04 > G example2.cpp > ...

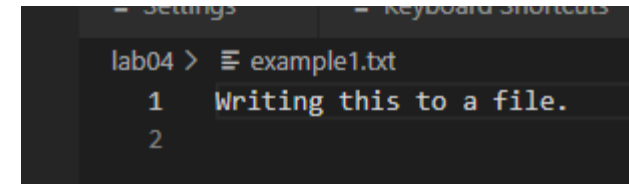
```
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  using namespace std;
5
6  int main()
7  {
8      ifstream myFile ("example1.txt");
9      string s;
10
11     cout << "Read File: " << endl;
12
13     if(myFile.is_open()) {
14         while(getline(myFile, s))
15             cout << s << endl;
16     }
17
18     return 0;
19 }
20
```

Executing task: cmd /C C:\Users\AnBC\Documents\D

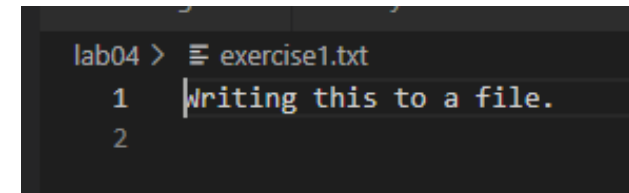
Read File:
Writing this to a file.

실습 1 - copy

- exercise1.cpp를 생성 후, example1.txt의 내용을 읽어서 exercise1.txt에 내용을 copy하자
- 사용하는 변수명은 무관
- 상대 경로 혹은 절대 경로 무관
- Terminal에 출력 내용 X
- example1.txt와 exercise1.txt의 내용만 file을 직접 열어 확인
- close()를 사용하여 file을 close
- example1.txt는 example1 – write에서 사용한 text file



```
lab04 > cat example1.txt
1 Writing this to a file.
2
```



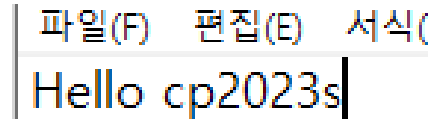
```
lab04 > cat exercise1.txt
1 Writing this to a file.
2
```

실습 1 - copy

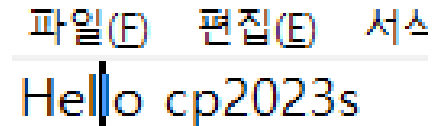
```
lab04 > exercise1.cpp > ...
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  using namespace std;
5
6  int main()
7  {
8      ifstream From ("example1.txt");
9      ofstream To ("exercise1.txt");
10     string s;
11
12     if(From.is_open()) {
13         while(getline(From, s))
14             To << s << endl;
15     }
16
17     From.close();
18     To.close();
19
20     return 0;
21 }
22
```

I/O Stream Position

- 각 IO stream object들은 read나 write을 하면 I/O 수행할 position이 바뀜
- 우리가 생각하는 cursor와 동일한 기능
- read하고 write할 위치를 바꿀 수 있음
- ifstream은 read⁰이므로 ***get position*** , ofstream은 write⁰이므로 ***put position***



파일(F) 편집(E) 서식(S)
Hello cp2023s



파일(F) 편집(E) 서식(S)
Hello cp2023s

I/O Stream Position

- **tellg()** / **tellp()** = 현재 get position / put position 값을 return ← return type == *streampos*
정수라 생각해도 무방
- **seekg(position)** / **seekp(position)** = 현재 get position / put position 값을 *position*으로 변경
- **seekg(offset, direction)**, **seekp(offset, direction)** = direction 에서 offset 만큼 옮김
- *offset* = byte 단위 정수 (음수라면 반대 방향)

• *direction* =

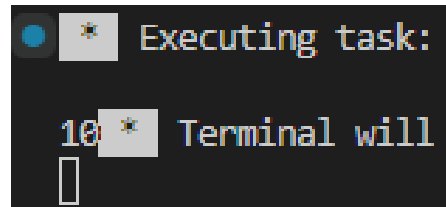
ios::beg	offset counted from the beginning of the stream
ios::cur	offset counted from the current position
ios::end	offset counted from the end of the stream

Example 3 – File size

- example3.txt를 ofstream으로 open하여 해당 file에 “0123456789”를 write 하고 close 하자
- 해당 file을 ifstream으로 open한 후, 현재 position을 streampos 변수에 할당 (tellg())
- seekg()를 이용해 file의 끝 위치로 get position을 옮김
- 현재 position을 streampos 변수에 할당 (tellg())
- 두 streampos 변수의 차를 이용해 file의 크기를 계산

Example 3 – File size

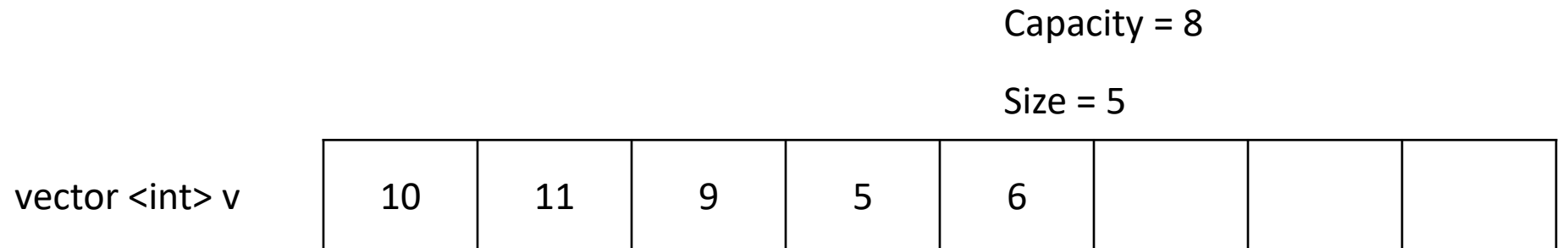
```
lab04 > G example3.cpp > ...
1  #include <iostream>
2  #include <fstream>
3
4  using namespace std;
5
6  int main()
7  {
8      ofstream myFile ("example3.txt");
9      ifstream myFile2;
10     streampos begin, end;
11
12     myFile << "0123456789";
13     myFile.close();
14
15     myFile2.open("example3.txt");
16
17     begin = myFile2.tellg();
18     myFile2.seekg(0, ios::end);
19     end = myFile2.tellg();
20
21     myFile2.close();
22
23     cout << end - begin << endl;
24
25     return 0;
26 }
```

A terminal window with a dark background. The title bar says "Executing task:". The first line of output is "10 * Terminal will". Below this line, there is a small white rectangular box, possibly a cursor or a placeholder.

Executing task: 10 * Terminal will

Vector

- 배열과 비슷 / 대신 Memory 관리가 편함
- Vector는 생성 후 Capacity가 **가변** / 배열은 고정
- Capacity vs. Size
 - Capacity = Vector가 담을 수 있는 원소의 개수
 - Size = 현재 Vector가 갖고 있는 원소의 수

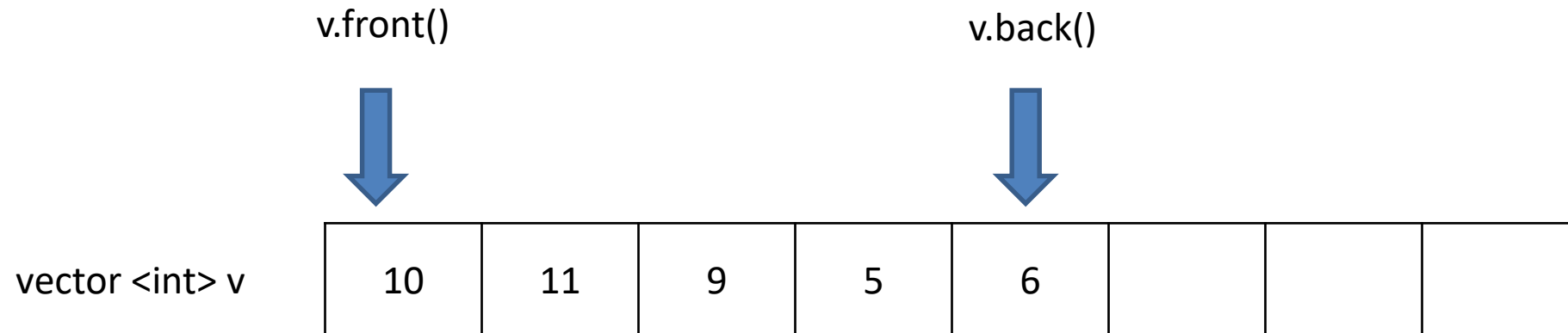


Vector

- 생성자
 - `vector<int> v;` ➔ 비어있는 vector `v`를 생성
 - `vector<int> v = { ... };` ➔ 원하는 값으로 초기화
 - `vector<int> v(5);` ➔ 0으로 초기화 된 5개 원소의 vector `v`를 생성
 - `vector<int> v(5, 2);` ➔ 2로 초기화 된 5개 원소의 vector `v`를 생성
 - `vector<int> v2(v1);` ➔ vector `v1`을 copy해 vector `v2`를 생성
- `v.at(idx)` ← `idx` 번째 원소를 참조 // 접근 가능 범위를 확인
- `v[idx]` ← `idx` 번째 원소를 참조 // 접근 가능 범위 확인 X ← 속도 더 빠름

Vector

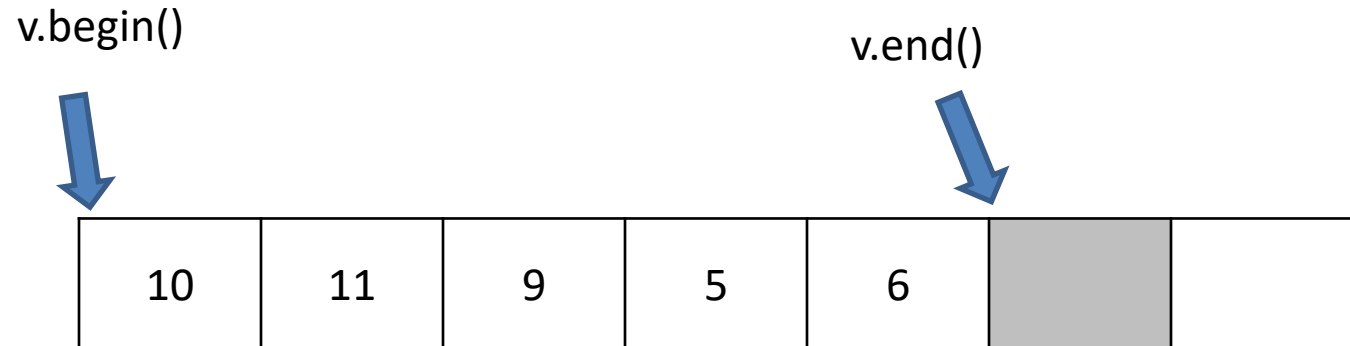
- `v.front()` : `v`의 첫번째 원소를 참조
- `v.back()` : `v`의 마지막 원소를 참조
- `v.clear()` : 모든 원소를 제거 / 원소만 제거되고 Memory는 그대로 / `Size = 0`, `Capacity` 그대로
- `v.size()` : `v`의 `size` 반환
- `v.capacity()` : `v`의 `capacity` 반환



Vector

- Iterator = 순환자 / 주소 값을 갖고 있을 수 있음
 ← pointer와 비슷하지만 객체 순환을 쉽게 할 수 있도록
- v.begin() : v의 첫 번째 원소의 iterator를 반환
- v.end() : v의 마지막 원소 + 1의 iterator를 반환

```
vector<int>::iterator itr;  
for (itr = v.begin(); itr != v.end(); ++itr) {  
    cout << *itr << endl;  
}
```



Example 4

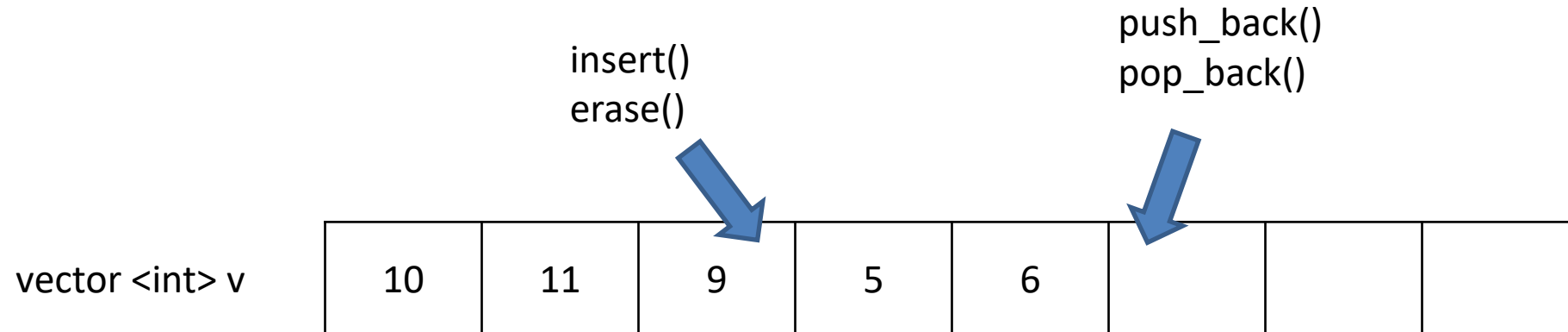
- vector header file을 include 한 후, 원하는 값으로 초기화 된 vector v를 생성
- iterator를 이용해 vector v의 원소를 출력해보고 v의 size를 출력해보자
- v의 capacity도 출력해보자

```
lab04 > example4.cpp > main()
1  #include <iostream>
2  #include <vector>
3
4  using namespace std;
5
6  int main()
7  {
8      vector<int> v = {10, 11, 9, 5};
9      vector<int>::iterator itr;
10
11     for(itr = v.begin(); itr != v.end(); ++itr) {
12         cout << *itr << endl;
13     }
14
15     cout << "size: " << v.size() << endl;
16     cout << "capacity: " << v.capacity() << endl;
17
18     return 0;
19 }
```

```
10
11
9
5
size: 4
capacity: 4
```

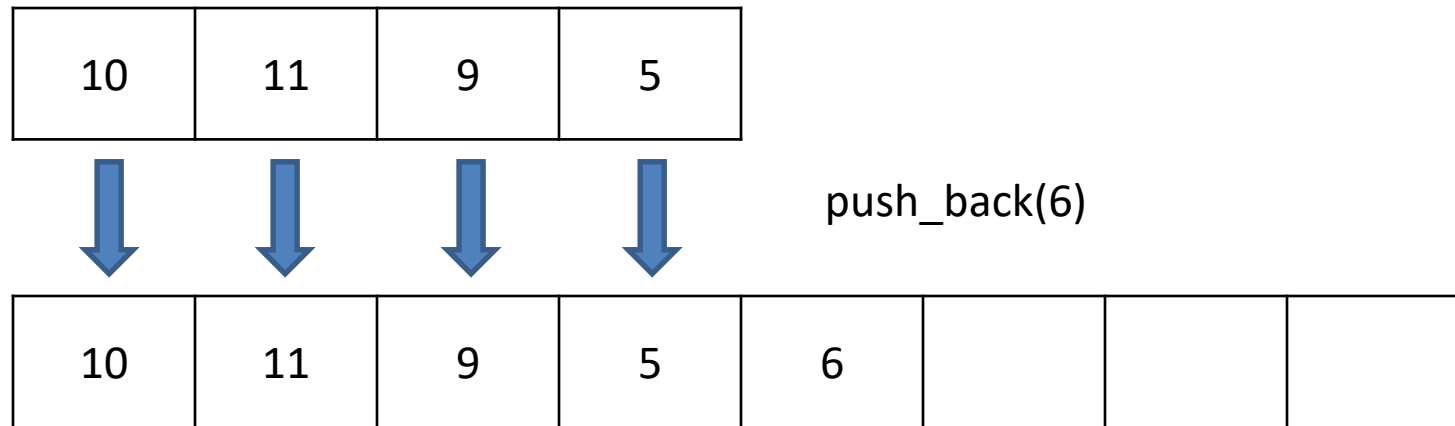
Vector

- `push_back(data)` = Vector의 맨 끝에 원소를 삽입
- `pop_back()` = Vector의 맨 끝 원소를 꺼냄
- `insert(loc, data)` = 원하는 location에 data를 삽입
- `erase(loc)` = 원하는 location의 data를 삭제



Vector

- 만약 `push_back()`을 진행하다가 초기에 만들었던 Capacity를 초과 한다면
 - Capacity가 두배로 늘어나며 새로운 Memory Location에 전부 copy되어 들어 감
- Capacity가 늘어난 후에 `pop_back()`을 한다 하여도 Size만 변할 뿐 Capacity는 그대로



실습 2

- 4개의 원소가 1로 초기화 된 vector v를 선언하자
- 현재 vector v의 capacity와 size를 출력해보자
- vector v의 시작 주소를 출력해보자 (C에서의 주소 접근과 동일)
- push_back()을 통해 vector v에 5 하나를 삽입하자
- 현재 vector v의 capacity, size와 시작 주소를 출력해보자
- pop_back()을 한 후, size, capacity와 시작 주소를 출력해보자

실습 2

```
int main()
{
    vector<int> v = {10, 11, 9, 5};

    cout << "Capacity: " << v.capacity() << endl;
    cout << "Size: " << v.size() << endl;
    printf("%p \n", &v[0]);
    printf("----- \n");

    cout << "After push_back()" << endl;
    v.push_back(6);

    cout << "Capacity: " << v.capacity() << endl;
    cout << "Size: " << v.size() << endl;
    printf("%p\n", &v[0]);

    printf("----- \n");
    cout << "After pop_back()" << endl;
    v.pop_back();
    printf("%p\n", &v[0]);
    cout << "Capacity: " << v.capacity() << endl;
    cout << "Size: " << v.size() << endl;

    return 0;
}
```

```
Capacity: 4
Size: 4
00711398
-----
After push_back()
Capacity: 8
Size: 5
007113B0
-----
After pop_back()
007113B0
Capacity: 8
Size: 4
```

* Terminal will b

Exercise 2

Exercise 2

- 23/4/12 ~ 23/4/18 23:59
- File I/O & Vector
- 1문제
 - 저장된 점수들을 A, B, C, D, fail 성적 범위 파일로 나누어 저장하고 싶다
 - 만들고 싶은 file의 이름이 filename.txt에 들어있음
 - 각 점수들은 score.txt에 들어 있음

Exercise 2

```
ex2 > ≡ filename.txt
1  A_score.txt
2  B_score.txt
3  C_score.txt
4  D_score.txt
5  fail_score.txt

ex2 > ≡ score.txt
1  41
2  67
3  34
4  0
5  69
6  24
7  78
8  58
9  62
10 64
11 5
12 45
13 81
14 27
15 61
16 91
17 95
```

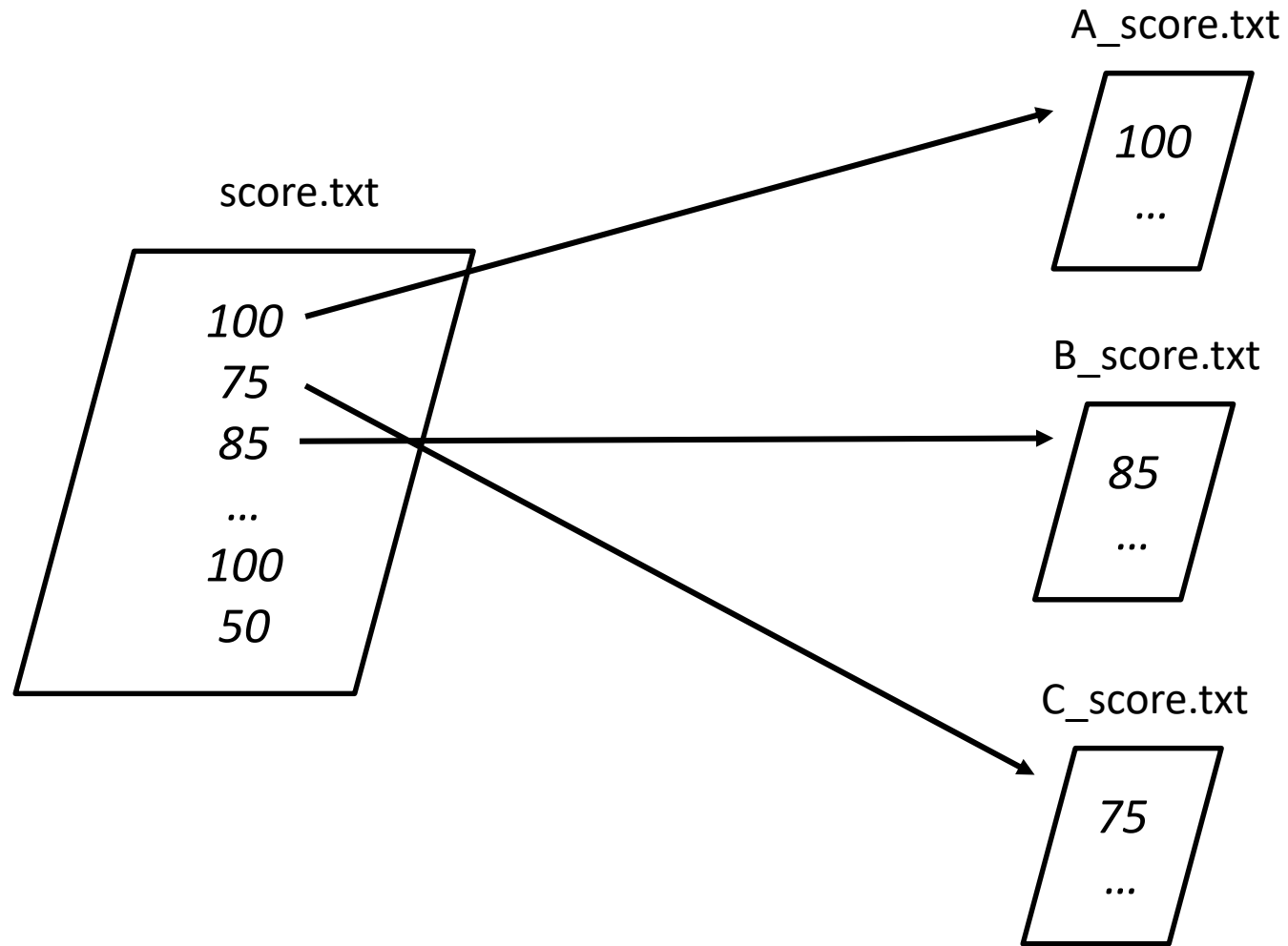


```

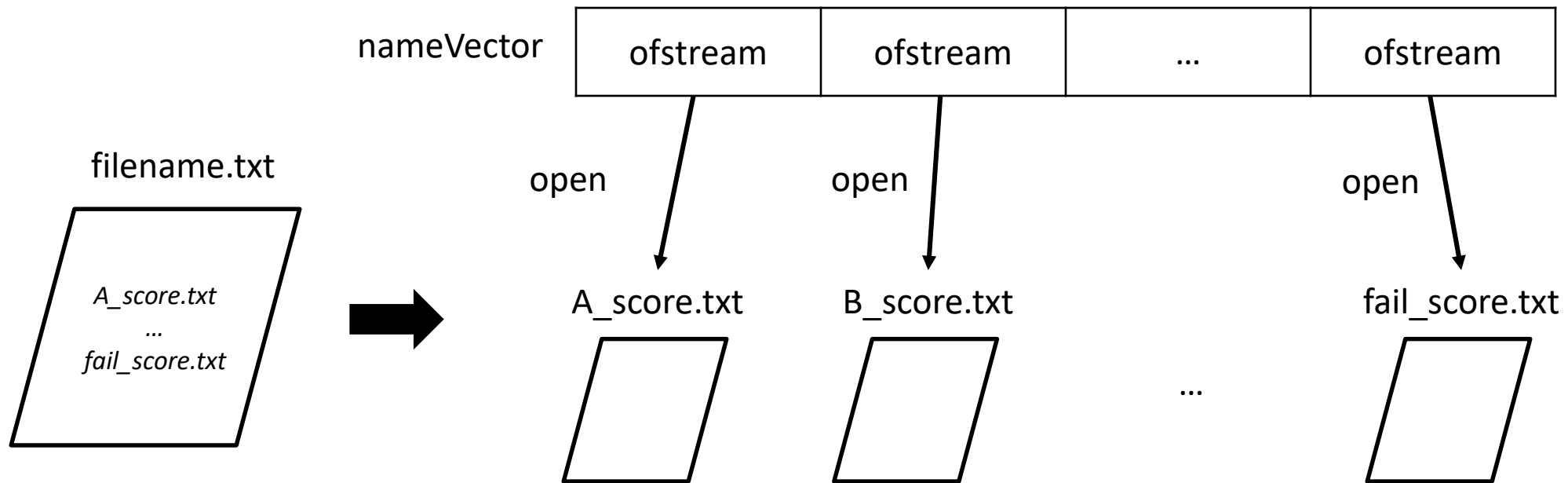
A_score 2023-04-10 오후 6:35 텍스트 문서
B_score 2023-04-10 오후 6:35 텍스트 문서
C_score 2023-04-10 오후 6:35 텍스트 문서
D_score 2023-04-10 오후 6:35 텍스트 문서
fail_score 2023-04-10 오후 6:35 텍스트 문서

ex2 > ≡ A_score.txt
1  91
2  95
3  91
4  92
5  95
6  99
7  94
8  90
9
```

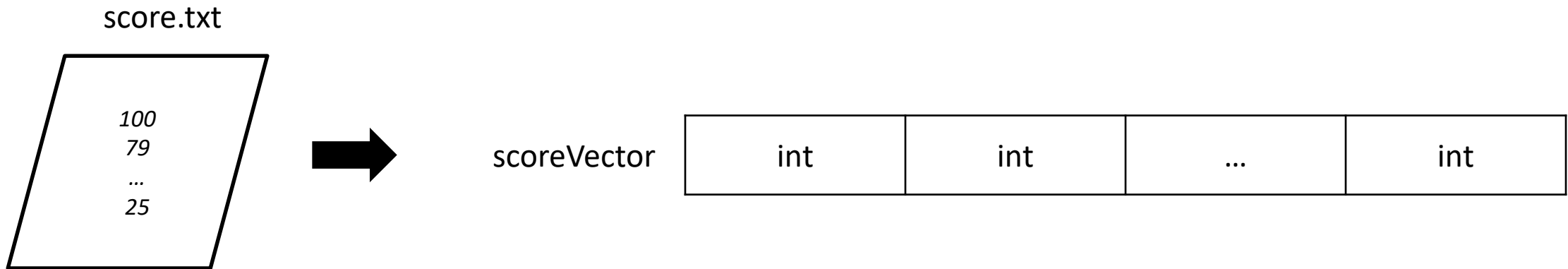
Exercise 2



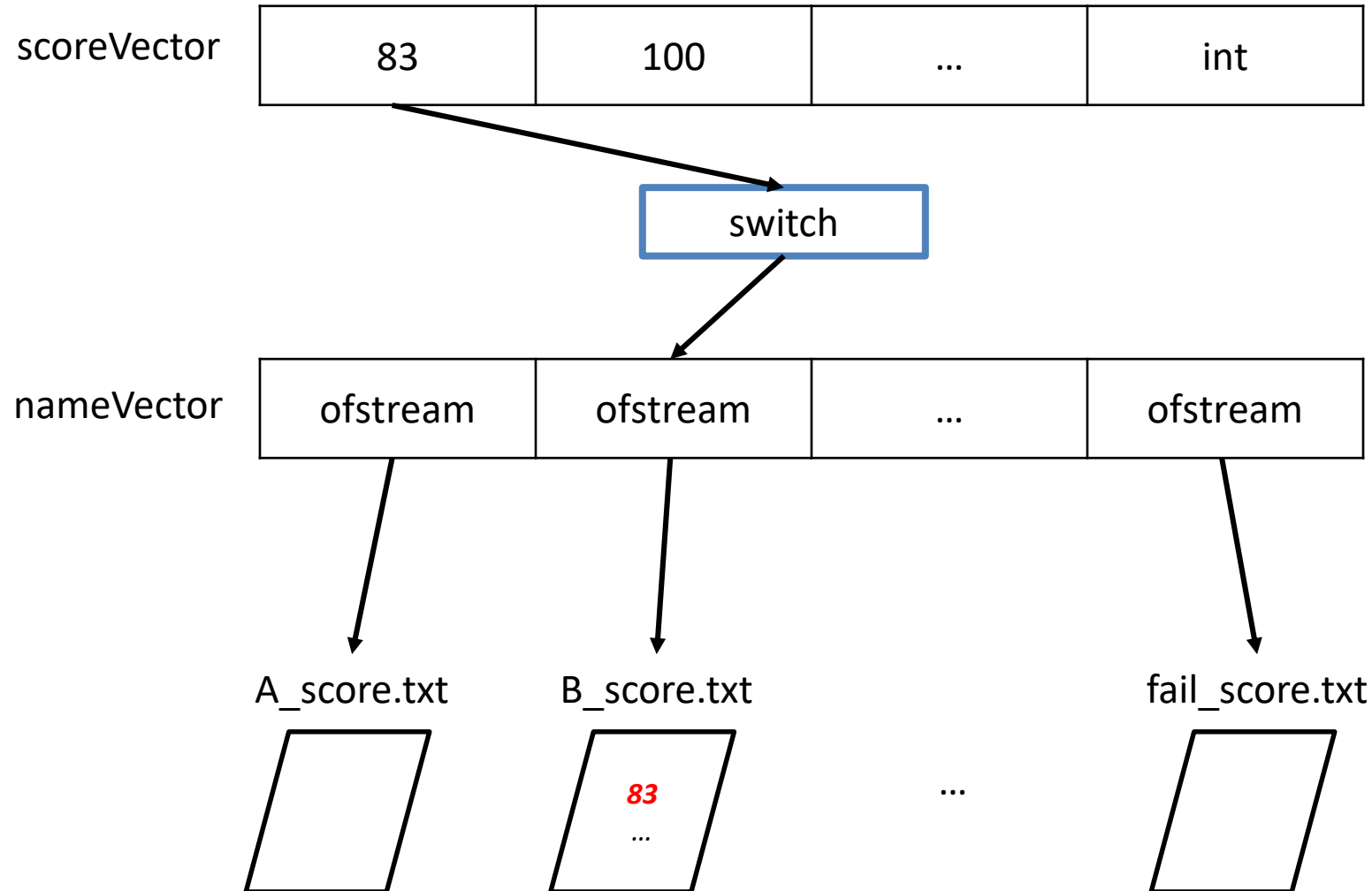
Exercise 2



Exercise 2



Exercise 2



Exercise 2

- `fstream`, `vector` library를 이용
- `filename.txt`를 한 줄씩 읽으며, 읽은 line을 file name으로 하는 file을 open해야 한다
- `filename.txt` 안에 있는 file name 수 만큼의 `ofstream`의 `vector`를 생성
- `score.txt`를 읽고 해당 시험 점수들을 `int`의 `vector`에 할당
- 해당 `score vector`를 조건에 맞춰 `ofstream vector`가 open한 file에 write

Exercise 2

- openGradeFiles() = filename.txt 안의 string을 이용해 file들을 create 및 open
- readScores() = scores.txt 안의 점수들을 읽음
- writeScores() = 점수들을 적절한 file에 write함
- closeInputFile() = input file stream을 close
- closeGradeFiles() = open한 grade file들을 모두 close 함

Exercise 2

```
ex2 > G+ EX2.cpp > openGradeFiles(..., ...)
1  #include <iostream>
2  #include <...> // - 각종 File IO를 위한 library를 include
3  #include <...> // - vector를 위한 library를 include
4  #include <string>
5
6  using namespace std;
7
8  void openGradeFiles(...);
9  void readScores(...);
10 void writeScores(...);
11
12 void closeInputFile(...);
13 void closeGradeFiles(...);
```

Exercise 2

```
15 void openGradeFiles(... filenameInput, ... nameVector) {           // 변수명은 수정하지 않고 진행
16     ... nameItr;                                                    // - nameVector를 위한 iterator 변수
17     ... tmpString;                                                  // - line read를 위한 string 변수
18
19     for(nameItr = ... ; nameItr != ...; ...) {                      // - iterator를 이용해 nameVector를 처음부터 끝까지 접근
20         if(!getline(*..., tmpString)) {
21             printf("Error: getline error\n");
22             exit(1);
23         }
24
25         ...                                                          // - iterator와 tmpString을 이용해 file create 및 open
26     }
27 }
28
29 void readScores(... inputFile, ... scoreVector) {                  // 변수명은 수정하지 않고 진행
30     ... tmpString;                                                  // - line read를 위한 string 변수
31
32     while(getline(*..., tmpString)) {                               // - inputFile을 한 줄 씩 읽어옴
33         ...                                                         // - 읽은 line을 stoi()를 이용해 scoreVector 맨 뒤에 삽입
34     }
35 }
```

Exercise 2

```
37 void writeScores(... scoreVector, ... nameVector) {           // 변수명은 수정하지 않고 진행
38     ... scoreItr;                                           // - scoreVector를 위한 iterator
39
40     for(scoreItr = ...; scoreItr != ...; ...) {             // - iterator를 이용해 scoreVector를 처음부터 끝까지 접근
41         switch (...)                                         // - scoreVector의 원소 값을 확인
42         {                                                    /* !!!** case 내의 ...은 범위를 표시하는 것이므로 수정하지 않습니다 **!! */
43             case 90 ... 100:
44                 ... << ... << "\n";                          // - 첫 번째 output file stream ("A_score.txt")에 write
45                 break;
46             case 80 ... 89:
47                 ... << ... << "\n";                          // - 두 번째 output file stream ("B_score.txt")에 write
48                 break;
49             case 70 ... 79:
50                 ... << ... << "\n";                          // - 세 번째 output file stream ("C_score.txt")에 write
51                 break;
52             case 60 ... 69:
53                 ... << ... << "\n";                          // - 네 번째 output file stream ("D_score.txt")에 write
54                 break;
55             default:
56                 ... << ... << "\n";                          // - 다섯 번째 output file stream ("Fail_score.txt")에 write
57                 break;
58         }
59     }
60 }
```

Exercise 2

```
62 void closeInputFile(... inputFile) {           // 변수명은 수정하지 않고 진행
63     ...                                         // - inputFile을 close
64 }
65
66 void closeGradeFiles(... nameVector) {          // 변수명은 수정하지 않고 진행
67     ... nameItr;                               // - nameVector를 위한 iterator
68
69     for(nameItr = ...; nameItr != ...; ...) {   // - iterator를 이용해 nameVetor를 처음부터 끝까지 접근
70         ...                                     // - iterator를 이용해 output file stream을 close
71     }
72 }
```


Exercise 2

```
74  int main() {
75      ... tmpfilenameInput = ... (...);           // for "filename.txt"
76      ... tmpscoreInput = ... (...);              // for "score.txt"
77
78      ... tmpnameVector = ...                     // for output file stream vector
79      ... tmpscoreVector = ...                    // for integer vector
80
81      ...                                           // Fuction Calls
82      ...
83      ...
84
85      ...
86      ...
87      ...
88
89      return 0;
90  }
```