Chapter 11 - C++ 표준 라이브러리

생성일	@2025년 11월 23일 오전 1:56
태그	

01) 표준 라이브러리 구성과 사용법

표준 라이브러리 구성

- 프로그램 시 활용 가능한 **범용 라이브러리의 집합체**
  - 100여 개가 넘는 헤더 파일로 구성되었으며 꾸준히 업데이트
- 주요 기능
  - **입출력** : 입출력 스트링을 사용하여 파일, 키보드, 화면과의 상호 작용 지원
  - **문자열 처리** : 문자열 조작, 검색, 대, 소문자 변환 등의 기능
  - **컨테이너** : 벡터, 리스크, 큐, 스택 등의 자료 구조 제공
  - **알고리즘** : 검색, 정렬, 변환, 그래프 알고리즘 등을 제공
  - 기타 **유틸리티** : 다양한 도구와 유틸리티 함수를 제공, 작업을 단순화하고 생산성을 높임

표준 라이브러리 사용 방법

```
# include <파일_이름>
```

- 표준 라이브러리 헤더를 포함할 때는 **화살 괄호 < >**를 사용
  - 사용자가 작성한 헤더 파일은 **큰 따옴표 “ “**를 사용함

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

int main()
{
```

```
    cout << "Hello World\n";
    return 0;
}
```

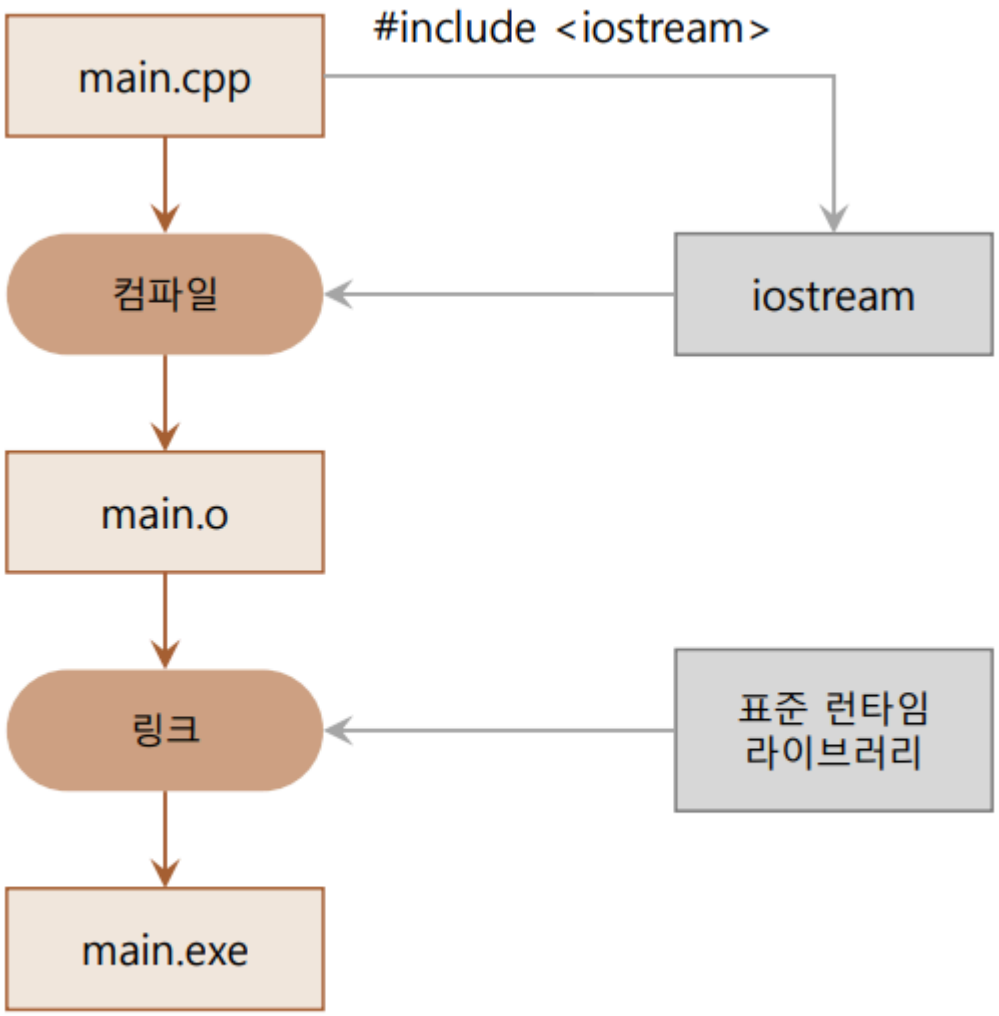


그림 표준 라이브러리 헤더 포함 과정

02) 문자열 라이브러리

표준 문자열 라이브러리 - std::string

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

using namespace std;

int main() {
    string str1("Hello");
    cout << str1 << endl;

    cout << str1[0] << endl;
    cout << str1[1] << endl;
    cout << str1[2] << endl;
    cout << str1[3] << endl;
    cout << str1[4] << endl;

    return 0;
}
```

실행 결과

```
Hello
H
e
l
l
o
```

- 객체를 생성할 때에 초깃값을 할당하거나 문자열을 직접 대입해 넣을 수도 있음

```
string str1("Hello");  // 생성자 호출로 초기화
string str1 = "Hello";  // 대입 연산으로 초기화
```

- 클래스 객체 문자 하나 하나에 직접 접근 가능

문자열 길이 구하기 — length, size

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

using namespace std;

int main() {
    string str1("Hello");
    cout << str1 << endl;

    cout << str1.length() << endl;
    cout << str1.size() << endl;

    return 0;
}
```

실행 결과

```
Hello
5
5
```

- **length 함수**만 **문자열의 길이**를 반환
- **size 함수**는 **객체가 차지하는 메모리의 크기**를 반환
    - 문자열의 길이가 필요할 때는 **length 함수**를 사용

빈 문자열인지 검사하기 — empty

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

using namespace std;

int main()
    string str1("");
    cout << str1 << endl;
```

```
    cout << std::boolalpha;
    cout << str1.empty() << endl; // true 또는 false 출력

    return 0;
}
```

실행 결과

```


true
```

- str1 변수에 문자열이 없으므로 empty 함수는 true를 반환

문자열 추가하기 — append

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

using namespace std;

int main() {
    string str1("Hello");
    str1.append(" World!");
    cout << str1 << endl;

    string str2("Hello");
    str2.append(" World!)", 6, 1);
    cout << str2 << endl;

    return 0;
}
```

실행 결과

```
Hello World!
Hello!
```

append 함수 사용법

```
문자열.append("추가할 문자열")
문자열.append("추가할_문자열", 문자열_시작_인덱스, 문자_개수)
```

- 추가할 문자열 중 일부만 추가할 수 있음
  - 추가할 문자열 " World!"에서 6번 인덱스를 시작으로 1개 문자인 "!"만 추가

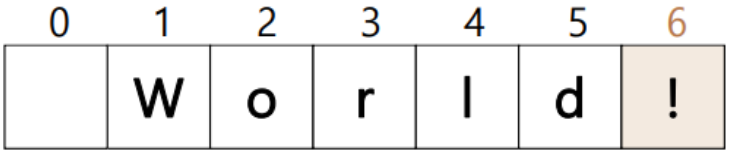


그림 append(" World!", 6, 1) 코드의 의미

문자열 찾기 — find

Do it! 실습 find 함수로 문자열 찾기

• ch11/string_find/string_find.cpp

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

using namespace std;

void check_found(string::size_type n) {
  if (n == string::npos) {
    cout << "not found" << endl;
  }
  else {
    cout << "found index: " << n << endl;
  }
}

int main() {
  string::size_type n;
  string str = "This is an example of a standard string.";

  // 문자열 시작 지점부터 "example" 탐색
  n = str.find("example");
  check_found(n);

  // 문자열 시작 지점부터 "is" 탐색
  n = str.find("is");
  check_found(n);

  // 문자열 내 index 위치 4부터 "is" 탐색
  n = str.find("is", 4);
  check_found(n);

  // 문자열 시작 지점부터 'h' 탐색
  n = str.find('h');
  check_found(n);

  // 문자열 시작 지점부터 'k' 탐색
  n = str.find('k');
  check_found(n);

  return 0;
}
```

문자열을 찾지 못했는지 검사

문자열의 크기를 나타내는 형식(부호 없는 정수)

find 함수는 대상 문자열을 찾지 못하면 정수 타입의 string::npos 라는 상수를 반환합니다

실행 결과

```
found index: 11
found index: 2
found index: 5
found index: 1
not found
```

find 함수 사용법

```
문자열.find(찾을_문자열);
문자열.find(찾을_문자);
문자열.find(찾을_문자열, 시작_위치);
```

이지스퍼블리싱

문자열 비교하기 — compare

compare 함수 사용법

```
대상_문자열.compare(비교할_문자열);
```

- **compare 함수**는 <u>두 문자열의 차이를 나타내는 정수를 반환</u>
  - **0** : 두 문자열이 완전히 같음
  - **양수** : 대상 문자열이 더 길거나 일치하지 않는 첫 번째 문자가 더 큼
  - **음수** : 대상 문자열이 더 짧거나 일치하지 않는 첫 번째 문자가 더 작음
- **예시**

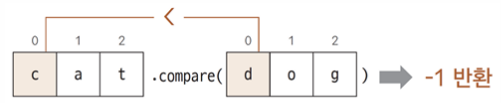


그림 cat, dog 문자열 비교

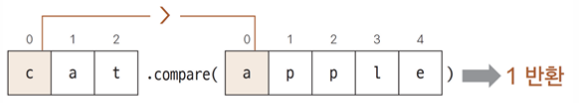


그림 cat, apple 문자열 비교

문자열 비교하기 — compare

Do it! 실습 compare 함수로 문자열 비교하기

• ch11/string_compare/string_compare.cpp

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

using namespace std;

void compare_result(int result) {
  if (result == 0) {
    cout << result << " = 두 문자열이 같음" << endl;
  }
  else if (result > 0) {
    cout << result << " = 대상 문자열이 더 길거나
                    일치하지 않는 첫 번째 문자가 더 큼" << endl;
  }
  else if (result < 0) {
    cout << result << " = 대상 문자열이 더 짧거나
                    일치하지 않는 첫 번째 문자가 더 작음" << endl;
  }
}

int main() {
  string s1 = "Hello";
  string s2 = "Hello";
  int result = s1.compare(s2);
  compare_result(result);
```

❶ s1, s2 문자열이 같을 때

```
  s1 = "Hello";
  s2 = "Hello World";
  result = s1.compare(s2);
  compare_result(result);
```

❷ s1 문자열의 길이가 더 짧을 때

```
  s1 = "cat";
  s2 = "dog";
  result = s1.compare(s2);
  compare_result(result);
```

❸ s1 문자열의 첫 번째 문자가 알파벳 순서상 먼저 나올 때

```
  s1 = "Hello World";
  s2 = "Hello";
  result = s1.compare(s2);
  compare_result(result);
```

❹ s1 문자열의 길이가 더 클 때

```
  s1 = "cat";
  s2 = "apple";
  result = s1.compare(s2);
  compare_result(result);
```

❺ s1 문자열의 길이가 더 짧지만, 일치하지 않는 첫 번째 문자가 알파벳 순서상 늦게 나올 때

```
  return 0;
}
```

실행 결과

```
0 = 두 문자열이 같음
-1 = 대상 문자열이 더 짧거나 일치하지 않는 첫 번째 문자가 더 작음
-1 = 대상 문자열이 더 짧거나 일치하지 않는 첫 번째 문자가 더 작음
1 = 대상 문자열이 더 길거나 일치하지 않는 첫 번째 문자가 더 큼
1 = 대상 문자열이 더 길거나 일치하지 않는 첫 번째 문자가 더 큼
```

Do it! 실습 관계 연산자로 문자열 비교

• ch11/string_compare_operator/string_compare_operator.cpp

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

using namespace std;

int main() {
  string s1 = "Hello";
  string s2 = "Hello";
  if (s1 == s2) {
   cout << "두 문자열 일치" << endl;
  }

  s1 = "Hello";
  s2 = "World";
  if (s1 != s2) {
    cout << "두 문자열 불일치" << endl;
  }

  return 0;
}
```

실행 결과

```
두 문자열 일치
두 문자열 불일치
```

- 문자열을 간단하게 비교할 때
  - compare 함수 대신 <, >, !=, == 등 관계 연산자 사용
  - 관계 연산 결과가 참이면 true, 거짓이면 false

- **문자열을 간단하게 비교할 때**
  - compare 함수 대신 **<, >, ≠, ==** 를 관계 연산자 사용
  - 관계 연산 결과가 **참**이면 **true**, **거짓**이면 **false**

문자열 교체하기 — replace

- replace는 문자열의 일부를 다른 문자열로 교체하는 함수

표 replace 함수 종류

문자열 종류	함수 원형
문자열 (string)	string& replace(size_t pos, size_t len, const string &str); string& replace(const_iterator i1, const_iterator i2, const string &str);
부분 문자열 (substring)	string& replace(size_t pos, size_t len, const string &str, size_t subpos, size_t sublen = npos);
C 언어 스타일 문자열 (c-string)	string& replace(size_t pos, size_t len, const char *s); string& replace(const_iterator i1, const_iterator i2, const char *s);

Do it! 실습 replace 함수로 문자열 교체하기

• ch11/string_replace/string_replace.cpp

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

using namespace std;

int main() {
  string str = "Hello, world!";
  cout << "Original string: " << str << std::endl;

  // 문자열 일부분을 교체
  str.replace(7, 5, "C++");
  cout << "Replaced string: " << str << endl;
  return 0;
}
```

실행 결과

```
Original string: Hello, world!
Replaced string: Hello, C++!
```

- "Hello, world!" 문자열에서 7번 인덱스에 있는 'w'부터 5개의 문자열을 "C++" 문자열로 교체
  - 결과 : "world" 문자열이 "C++" 문자열 변경
- fine 함수와 replace 함수를 함께 사용

Do it! 실습 find 함수와 함께 사용한 replace 함수

• ch11/string_find_replace/string_find_replace.cpp

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

using namespace std;

int main() {
  // 문자열 생성
  string text = "The C++ programming language is one of the hardest
                 languages.";

  // 찾을 문자열과 교체할 문자열 정의
  string target = "hardest";
  string replacement = "most powerful";

  // 처음 등장하는 위치 찾기
  size_t pos = text.find(target);
  // 문자열 교체
  if (pos != string::npos) {
    text.replace(pos, target.length(), replacement);
    cout << "교체 후 문장: " << text << endl;
  }
  else {
    cout << target << " 을 찾을 수 없음" << endl;
  }

  return 0;
}
```

실행 결과

```
교체 후 문장: The C++ programming language is one of the most powerful
languages.
```

20

이지스퍼블리싱

와이드 문자열 — wstring

- C++11부터 **std:: wstring 와이드 문자열 형식** 제공
- **setlocale 함수**는 프로그램의 지역을 설정
  - **LC_ALL**은 모든 지역을 의미
- **유니코드 문자열을 초기화**
  - **접두어 L**은 문자열 리터럴이 와이드 문자열임을 표시
  - 문자열 출력 스트림인 **wcout**을 사용

Do it! 실습 wstring 활용하기

• ch11/wstring/wstring.cpp

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

using namespace std;

int main() {
  // 프로그램의 지역 설정
  setlocale(LC_ALL, "");

  // 유니코드 문자열 초기화
  wstring korString = L"안녕하세요";

  // 유니코드 문자열 출력
  wcout << korString << endl;

return 0;
}
```

실행 결과

```
안녕하세요
```

03) 파일 시스템

파일 시스템이란 무엇일까?

- 파일 시스템 : 데이터를 저장하고 관리하는 체계
- 파일 시스템은 OS에 따라 디렉터리, 파일 종류, 권한, 상태, 속성 등의 구조가 다름
  - OS마다 파일 시스템 프로그래밍에서 사용하는 헤더 파일과 사용 방법이 다름

파일 시스템 라이브러리

- **C++ 표준 라이브러리**는 파일 시스템과 관련된 다양한 형식과 함수를 OS에 범용적으로 지원
- **C++ 파일 시스템 라이브러리**는 std::filesystem 네임스페이스에 속해 있음

- 소스 파일에 **<filesystem> 헤더**를 포함

표 파일 시스템 라이브러리가 제공하는 주요 기능

구분	이름	설명
클래스	path	파일과 디렉터리 경로를 나타내는 클래스. 경로 조작이나 분석 지원
	directory_entry	디렉터리 내의 항목을 나타내는 클래스. 경로나 속성 정보 제공
	directory_iterator	디렉터리 내의 모든 항목을 순회하기 위한 반복자 클래스
	file_status	파일이나 디렉터리의 상태 정보를 나타내는 열거형 클래스
	file_time_type	파일의 시간 정보를 나타내는 file_time과 같은 형식
	space_info	디스크 공간에 대한 정보를 제공하는 클래스
열거형	perms	파일이나 디렉터리의 권한을 나타내는 열거형 상수 집합
	file_type	파일이나 디렉터리의 유형을 나타내는 열거형 상수 집합
함수	exists	주어진 경로에 파일이나 디렉터리가 존재하는지 확인
	is_directory	주어진 경로가 디렉터리인지 확인
	is_regular_file	주어진 경로가 일반 파일인지 확인
	create_directory	디렉터리 생성
	create_directories	경로에 지정된 디렉터리나 중간 디렉터리 생성
	remove	파일이나 디렉터리 삭제

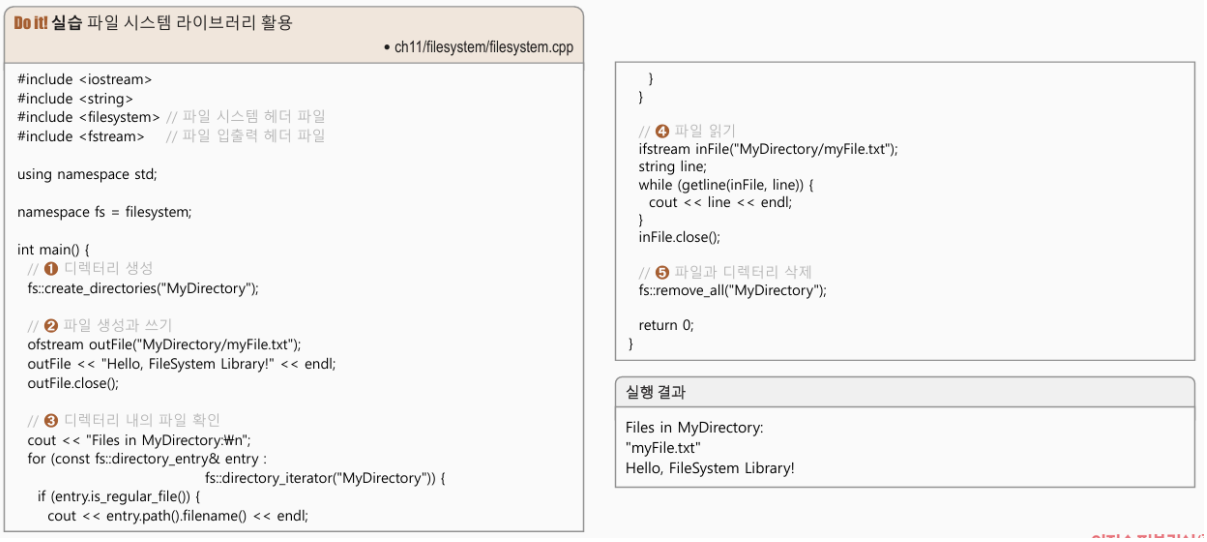
구분	이름	설명
함수	rename	파일이나 디렉터리의 이름 변경
	copy	파일이나 디렉터리 복사
	copy_file	파일을 복사
	copy_directory	디렉터리를 복사
	copy_symlink	심벌릭 링크 복사
	file_size	파일의 크기 반환
	last_write_time	파일이나 디렉터리의 마지막 수정 시간 반환
	current_path	현재 작업 디렉터리의 경로 반환
	equivalent	두 경로가 같은 파일이나 디렉터리를 가리키는지 확인
	is_empty	주어진 디렉터리가 비었는지 확인
	remove_all	디렉터리와 하위 항목 모두 삭제
	resize_file	파일 크기 변경
	status	파일이나 디렉터리의 상태 정보 반환
	temp_directory_path	임시 디렉터리의 경로 반환

- **절대 경로와 상대 경로**
  - 컴퓨터 상의 모든 파일에는 해당 파일의 위치를 나타내는 고유 주소 → **경로**
  - **절대 경로** : 최상위 디렉터리인 루트에서 내가 원하는 파일까지의 전체 경로
    - 예) C:\, D:\ (윈도우 OS), / (리눅스 계열)
  - **상대 경로** : 현재 위치를 기준으로 한 경로
- **경로를 나타내는 path 객체**
  - 파일 시스템 라이브러리의 많은 함수는 **path 객체를 매개변수로** 입력 받음
- **파일 path 객체가 실제로 존재하는지 확인 : exists 함수**

exists 함수 원형

```
bool exists(const std::filesystem::path& p)
```

파일 시스템 활용하기



Do it! 실습 파일 시스템 라이브러리 활용

• ch11/filesystem/filesystem.cpp

```cpp
#include <iostream>
#include <string>
#include <filesystem> // 파일 시스템 헤더 파일
#include <fstream>    // 파일 입출력 헤더 파일

using namespace std;

namespace fs = filesystem;

int main() {
  // ❶ 디렉터리 생성
  fs::create_directories("MyDirectory");

  // ❷ 파일 생성과 쓰기
  ofstream outFile("MyDirectory/myFile.txt");
  outFile << "Hello, FileSystem Library!" << endl;
  outFile.close();

  // ❸ 디렉터리 내의 파일 확인
  cout << "Files in MyDirectory:₩n";
  for (const fs::directory_entry& entry :
                          fs::directory_iterator("MyDirectory")) {
    if (entry.is_regular_file()) {
      cout << entry.path().filename() << endl;
    }
  }

  // ❹ 파일 읽기
  ifstream inFile("MyDirectory/myFile.txt");
  string line;
  while (getline(inFile, line)) {
    cout << line << endl;
  }
  inFile.close();

  // ❺ 파일과 디렉터리 삭제
  fs::remove_all("MyDirectory");

  return 0;
}
```

실행 결과

```
Files in MyDirectory:
"myFile.txt"
Hello, FileSystem Library!
```

- **디렉터리 생성**
  - **create_directories 함수**는 지정된 경로에 디렉터리를 생성
  - 필요하면 **중간 단계의 디렉터리**도 함께 생성

```
fs::create_directories("path/to/MyDirectory");
```

  - **path와 to 디렉터리가 없으면 새로 생성**, 이미 있어도 오류가 발생하지 않음

- **파일 생성과 쓰기**

```
ofstream outFile("MyDirectory/myFile.txt");
outFile << "Hello, FileSystem Library!" << endl;
outFile.close();
```

  - **outFile.close 함수**를 호출하는 이유는 파일을 제대로 닫기 위함
  - ofstream의 소멸자가 파일을 자동으로 닫아 주지만, **명시적으로 close 함수 하는 것이 해석**이다.

- **디렉터리 탐색**

```
cout << "Files in MyDirectory:\n";
 for (const fs::directory_entry& entry : fs::directory_i
terator("MyDirectory")) {
```

```
 if (entry.is_regular_file()) {
 cout << entry.path().filename() << endl;
 }
 fs::create_directories("path/to/MyDirectory");
 }
}
```

- **디렉터리 반복자 (directory_iterator)**를 생성
  - 해당 디렉터리의 파일과 디렉터리 정보를 제공
- **범위 기반 for문**으로 모든 원소를 대상으로 탐색 반복
  - **entry**는 각 파일이나 디렉터리에 대한 참조
- 디렉터리가 아닌 파일인지 확인하고 현재 파일의 경로에서 파일 이름만 출력
  - **entry.path 함수**는 파일의 전체 경로, **filename 함수**는 파일 이름만 추출

- **파일 읽기**

```
ifstream inFile("MyDirectory/myFile.txt");
 string line;
 while (getline(inFile, line)) {
 cout << line << endl;
 }
 inFile.close();
```

  - **ifstream**은 파일을 읽기 위한 입력 스트림을 제공
    - **getline 함수**로 파일에서 한 줄씩 읽어 line에 저장한 후 화면에 출력하는 동작을 파일의 끝까지 반복
  - **inFile.close 함수**를 호출하여 파일을 닫음, 파일 자원을 해제하고 접근을 종료

- **파일 삭제**

```
fs::create_directories("path/to/MyDirectory");
```

  - ***지정한 디렉터리와 그 안에 포함된 모든 요소를 삭제***
    - 하위 디렉터리나 파일이 있더라도 재귀하는 방법으로 모두 삭제 (완전 삭제이므로 주의)

04) 기타 유용한 함수

난수 생성

- **난수** : 정의된 범위에서 ***무작위로 추출된 임의의 수***
  - 난수는 그 다음에 나올 값을 확신할 수 없어야 함. 시드라는 시작 숫자를 이용해 무작위성을 증가함
    - 시드 값으로 현재 시각을 사용하는 경우가 많음

- **C와 C++ 언어에서는 난수를 생성하는 rand와 srand 함수**
  - rand 함수는 난수 생성 패턴을 한 개로 설정, srand 함수는 난수 생성 패턴을 여러 개로 설정
  - ***난수의 범위가 0 ~ 32,767로 넓지 않아서*** 난수가 불균등하게 분포

- **C++11부터는 고품질의 난수 생성기와 분포 클래스를 제공**
  - 난수의 형식, 범위, 분포와 형태 등을 세세하게 조절
  - 다음 코드는 **<random>** 헤더 파일에 있는 **std::mt19937을 이용**하는 예
    - mt19937은 32bit 버전, 64bit 버전 mt19937_64

Do it! 실습 난수 생성하기

• ch11/random/random.cpp

```
#include <iostream>
#include <random>

using namespace std;

int main()
{
  mt19937_64 mt_rand;

  for (int i = 0; i < 10; i++) {
    cout << mt_rand() << endl;
  }

  return 0;
}
```

실행 결과

```
14514284786278117030
4620546740167642908
13109570281517897720
17462938647148434322
355488278567739596
7469126240319926998
4635995468481642529
418970542659199878
9604170989252516556
6358044926049913402
```

10개 모두 임의의 수가 생성되었습니다. 그런데, 이 예제를 반복해서 실행해 보면 늘 동일한 10개의 임의의 수가

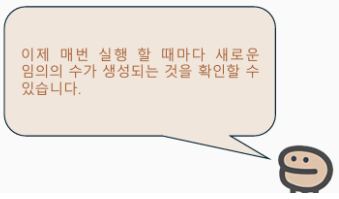
Do it! 실습 시드값으로 난수 생성하기

• ch11/chrono/chrono.cpp

```
#include <iostream>
#include <random>
#include <chrono>

using namespace std;

int main()
{
  // 시드값 사용
  auto curTime = chrono::system_clock::now();
  auto duration = curTime.time_since_epoch();
  auto millis =
          chrono::duration_cast<chrono::milliseconds>(duration).count();

  mt19937_64 mt_rand(millis);

  for (int i = 0; i < 10; i++) {
    cout << mt_rand() << endl;
  }

  return 0;
}
```

실행 결과

```
15679643833092956256
12842249489731865397
1730214222799370782
1030118958332042119
18147668297370799075
7404339794962184967
5829574859668400879
10697024794190343412
6673049286310623441
10615868138674902293
```



- **하드웨어 엔트로피 사용** : 시스템에서 발생하는 무작위성의 정도
  - 마우스 움직임, 커서 위치, 키보드 입력, 디스크 I/O 등을 활용하여 엔트로피 수집
  - **random_device**는 대체로 mt19937 엔진보다 느림

    → mt10037의 시드로 random_device 값을 사용

Do it! 실습 하드웨어 엔트로피로 난수 생성하기

• ch11/random_device/random_device.cpp

```
#include <iostream>
#include <random>

using namespace std;

int main()
{
  random_device rng;

  for (int i = 0; i < 10; i++) {
    auto result = rng();
    cout << result << endl;
  }

  return 0;
}
```

실행 결과

```
3499157824
2033147778
4023491842
553450092
406820767
3272902983
991870752
3827992927
1136119549
2107782506
```

- **<random>**에는 random_device 외에도 3가지 난수 생성 엔진을 제공
  - **linear_congruential_engine** : 선형 합동 난수 엔진
  - **mersenne_twiseter_engine** : 메르센 트위스터 난수 엔진
  - **subtract_with_carry_engine** : 감산 캐리 난수 엔진

수학 함수

- **<cmath> 헤더 파일**을 이용하면 삼각 함수, 지수 로그. 로그 함수 같은 수식을 쉽게 사용

표 자주 사용하는 수학 함수

함수	설명
abs	절댓값 반환
sqrt	제곱근 반환
pow	거듭제곱 계산
exp	지수 함수 (e^x) 계산
log	자연 로그 계산
sin, cos, tan	삼각 함수 계산
asin, acos, atan	역삼각 함수 계산
ceil	올림 계산
floor	내림 계산
rund	반올림 계산

Do it! 실습 수학 함수 활용하기

• ch11/cmath/cmath.cpp

```
#include <iostream>
#include <cmath>
#include <numbers>

using namespace std;

namespace fs = filesystem;

int main() {
  double x = 2;
  double y = 3;
  cout << "x = " << x << ", " << "y = " << y << endl;
  cout << "pow(x, y) = " << pow(x, y) << endl;
  cout << "sqrt(x) = " << sqrt(x) << endl;
  cout << "log(x) = " << log(x) << endl;
  cout << endl;

  x = 2.847;
  y = -3.234;
  cout << "x = " << x << ", " << "y = " << y << endl;
  cout << "ceil(x) = " << ceil(x) << ", ceil(y) = " << ceil(y) << endl;
  cout << "floor(x) = " << floor(x) << ", floor(y) = " << floor(y) << endl;
  cout << "round(x) = " << round(x) << ", round(y) = " << round(y) << endl;
  cout << "abs(x) = " << abs(x) << ", abs(y) = " << abs(y) << endl;
  cout << endl;
  cout << "PI = " << numbers::pi << endl;
  cout << "sin(PI/3) = " << sin(numbers::pi / 3) << endl;
  cout << "cos(PI/3) = " << cos(numbers::pi / 3) << endl;
  cout << "tan(PI/3) = " << tan(numbers::pi / 3) << endl;

  return 0;
}
```

원주율 파이(π)를 나타내는 상수 numbers::pi는 C++20부터 사용할 수 있습니다.

실행 결과

```
x = 2, y = 3
pow(x, y) = 8
sqrt(x) = 1.41421
log(x) = 0.693147
x = 2.847, y = -3.234
ceil(x) = 3, ceil(y) = -3
floor(x) = 2, floor(y) = -4
round(x) = 3, round(y) = -3
abs(x) = 2.847, abs(y) = 3.234
PI = 3.14159
sin(PI/3) = 0.866025
cos(PI/3) = 0.5
tan(PI/3) = 1.73205
```

복사 함수

- **7-3절 생성자와 소멸자 복습; 복사 생성자에서 배웠던 얕은 복사. 깊은 복사**
  - **얕은 복사** : 주솟값을 복사

- **깊은 복사** : 실제 값을 새로운 메모리 공간에 복사

  → 의도에 따라 적절하게 사용해야 함

- **깊은 복사를 반복문으로 직접 구현**
  - 객체에 포함된 모든 원소를 하나하나 복사

    → 코드도 길고, 양에 따라 실행 시간도 오래 걸림
  - 표준 라이브러리에서는 이를 보완하고자 ***깊은 복사 함수 copy*** 제공

    exists 함수 원형

```
template <class _InIt, class _OutIt>
 _CONSTEXPR20 _OutIt copy(_InIt _First, _InIt _Last,
_OutIt _Dest)
```

    - _First부터 _List 전 까지의 모든 원소를 _Dest부터 시작하는 곳에 복사

Do it! 실습 copy 함수 활용하기

• ch11/copy/copy.cpp

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

// 사람 정보를 담는 구조체
struct Person {
  string name;  // 이름
  int age;      // 나이
  float height; // 키
  float weight; // 몸무게
};

// 벡터에 저장된 사람 정보 출력 함수
void print_person_all(vector<Person>& vec) {
  for (vector<Person>::iterator it = vec.begin(); it != vec.end();
                                                  it++) {
    cout << "이름: " << it->name << "Wt > " << "나이: " << it->age
         << ", " << "키: " << it->height << ", " << "몸무게: "
         << it->weight << endl;
  }
}

int main() {
  // Person 구조체 배열 생성
  Person p[5] = {
      {"Brain", 24, 180, 70},
      {"Jessica", 22, 165, 55},
      {"James", 30, 170, 65},
      {"Tom", 12, 155, 46},
      {"Mary", 18, 172, 62}
  };

  // from_vector에 Person 배열 순서대로 넣기
  vector<Person> from_vector;
  from_vector.push_back(p[0]);
  from_vector.push_back(p[1]);
  from_vector.push_back(p[2]);
  from_vector.push_back(p[3]);
  from_vector.push_back(p[4]);

  // from_vector 출력
  cout << "-----from_vector-----" << endl;
  print_person_all(from_vector);
  cout << endl;

  // to_vector에 from_vector의 원소를 '깊은 복사' 수행
  vector<Person> to_vector;
  copy(from_vector.begin(), from_vector.end(),
                            back_inserter(to_vector));

  // 복사 후 to_vector 출력
  cout << "-----to_vector-----" << endl;
  print_person_all(to_vector);
  cout << endl;

  // from_vector의 첫 번째 원소 수정
  from_vector[0].name = "Chris";
  from_vector[0].age = 5;
  from_vector[0].height = 110;
  from_vector[0].weight = 20;

  // 수정 후 from_vector 출력
  cout << "-----from_vector-----" << endl;
  print_person_all(from_vector);
  cout << endl;

  // to_vector 출력
  cout << "-----to_vector-----" << endl;
  print_person_all(to_vector);
  cout << endl;

  return 0;
}
```

실행 결과

```
-----from_vector-----
이름: Brain > 나이: 24, 키: 180, 몸무게: 70
이름: Jessica > 나이: 22, 키: 165, 몸무게: 55
이름: James > 나이: 30, 키: 170, 몸무게: 65
이름: Tom > 나이: 12, 키: 155, 몸무게: 46
이름: Mary > 나이: 18, 키: 172, 몸무게: 62

-----to_vector-----
이름: Brain > 나이: 24, 키: 180, 몸무게: 70
이름: Jessica > 나이: 22, 키: 165, 몸무게: 55
이름: James > 나이: 30, 키: 170, 몸무게: 65
이름: Tom > 나이: 12, 키: 155, 몸무게: 46
이름: Mary > 나이: 18, 키: 172, 몸무게: 62

-----from_vector-----
이름: Chris > 나이: 5, 키: 110, 몸무게: 20
이름: Jessica > 나이: 22, 키: 165, 몸무게: 55
이름: James > 나이: 30, 키: 170, 몸무게: 65
이름: Tom > 나이: 12, 키: 155, 몸무게: 46

-----to_vector-----
이름: Brain > 나이: 24, 키: 180, 몸무게: 70
이름: Jessica > 나이: 22, 키: 165, 몸무게: 55
이름: James > 나이: 30, 키: 170, 몸무게: 65
이름: Tom > 나이: 12, 키: 155, 몸무게: 46
이름: Mary > 나이: 18, 키: 172, 몸무게: 62
```