

C++ 프로그래밍 및 실습

CLI 환경 음원 서비스 프로그램

최종 보고서

제출일자 : 2024년 12월 22일

제출자명 : 박지환

제출자학번 : 184128

1. 프로젝트 목표

1) 배경 및 필요성

현재 수많은 음악 스트리밍 서비스와 음악 관리 프로그램이 있지만, 대부분 직관적인 GUI 중심의 대중적인 인터페이스를 갖추고 있습니다.

이는 일반 사용자들에게는 편리하지만, 개발자 또는 CLI 환경에서 작업을 하는 것이 익숙한 사용자들에게는 오히려 가벼운 CLI 환경에서 더 유연하게 음악을 관리할 수 있는 대안이 필요한 실정입니다.

특히 CLI 환경을 선호하는 사용자들은 대중적인 GUI 환경의 인터페이스 및 무거운 애플리케이션 대신, 익숙하고 개발자스러운 인터페이스와 명령어 기반의 조작이 더욱 직관적이고 흥미를 느낄 수 있다고 생각하였습니다.

이러한 이유로 본 프로젝트는 이런 사용자들을 위한 개발자 친화적인 음원 서비스 솔루션을 제안하고자 하였습니다.

독창적인 CLI 기반의 음원 서비스로, 사용자에게는 경량화된 환경에서 음악을 재생하고 관리할 수 있는 자유로움을 제공하며, 단순한 조작으로 자신만의 커스터마이징된 플레이리스트를 주체적으로 관리하고 음악을 즐길 수 있게 됩니다.

2) 프로젝트 목표

본 프로젝트의 목표는 CLI 환경에서 동작하는 음원 서비스 프로그램을 구현하여, 사용자가 음악을 추가/삭제/재생 및 관리할 수 있는 단순하고 직관적인 인터페이스를 제공하는 것입니다.

또한, 음악 재생 기능을 통합하여 CLI 환경 내에서 바로 음악을 감상할 수 있게 합니다.

3) 차별점

기존 음원 서비스와는 달리, 본 프로그램은 CLI 환경에 최적화되어 있어 가볍고 직관적이며, 최소한의 리소스만으로 효율적인 음악 재생이 가능합니다. 또한 CLI 사용자에게 친숙한 명령어 기반 인터페이스와 다양한 재생 옵션을 통해 음악 감상의 편리함을 극대화합니다.

또한 기본적인 기능에 대해 구현이 다 완료된 이후, 현재 발매된 음원이 아닌, AI를 기반으로 목소리를 변조하여 새로운 음악을 제작한 후 해당 음원을 사용자의 플레이리스트로 추가할 수 있는 기능 또한 추가할 예정입니다.

2. 기능 계획

1) 기능 1. 음악 파일 목록 읽어오기

- 설명: 특정 디렉토리(./songs/)에서 .mp3 확장자를 가진 음악 파일 목록을 읽어오는 기능을 제공합니다. 이를 통해 사용자에게 사용할 수 있는 음악의 리스트를 동적으로 생성하고 관리합니다.

(1) 세부 기능 1. 디렉토리 탐색

- 설명: 디렉토리 내 파일 목록을 읽고, .mp3 확장자를 가진 파일만 필터링하여 저장합니다. DIR과 dirent를 활용해 구현되며, 디렉토리 접근 오류를 처리할 수 있는 예외 처리를 포함합니다.

(2) 세부 기능 2. 파일명 토큰화

- 설명: 읽어온 파일명을 분석하여, 파일명에서 노래 제목, 아티스트, 재생 시간을 추출합니다. 파일명은 아티스트-노래 제목-재생 시간.mp3 형식으로 가정하며, - 및 .을 기준으로 나누어 데이터를 분리합니다.

(3) 세부 기능 3. 데이터 저장 및 관리

- 설명: 추출한 데이터를 map 자료구조를 사용해 인덱스 기반으로 저장합니다. 각 노래의 제목, 아티스트, 재생 시간은 songTitle, songArtist, songDuration 등으로 관리되며, 이후 사용자가 데이터를 빠르게 조회할 수 있도록 준비합니다.

2) 기능 2. 노래 목록 표시 및 선택

- 설명: 사용 가능한 노래 목록을 화면에 표시하고, 사용자가 목록에서 특정 노래를 선택할 수 있는 기능을 제공합니다. 선택된 노래는 이후 재생 또는 세부 정보 조회와 같은 작업을 진행합니다.

(1) 세부 기능 1. 노래 목록 출력

- 설명: map에 저장된 노래 데이터를 읽어와 사용 가능한 노래를 인덱스, 제목, 아티스트,

재생 시간과 함께 화면에 출력합니다. 리스트가 비어있을 경우, 알림 메시지를 표시합니다.

(2) 세부 기능 2. 사용자 입력 처리

- 설명: 사용자로부터 번호 입력을 받아, 유효한 번호일 경우 선택한 노래의 인덱스를 저장하고 이후 재생 상태로 진입합니다. 잘못된 입력값에 대해서는 예외 처리를 추가하여 프로그램의 안정성을 유지합니다.

(3) 세부 기능 3. 선택 결과 반영

- 설명: 사용자가 선택한 노래의 인덱스를 저장하고, 이를 기반으로 재생 화면으로 전환하거나 메뉴로 돌아가는 선택지를 제공합니다.

3) 기능 3. 음악 재생 및 정지

- 설명: 사용자가 선택한 노래의 정보를 출력하며, 재생/일시 정지, 이전/다음 곡 이동과 같은 제어 기능을 제공합니다. 재생 상태는 isPaused로 관리되며, 입력에 따라 상태를 동적으로 업데이트합니다.

(1) 세부 기능 1. 현재 노래 정보 출력

- 설명: 현재 재생 중인 노래의 제목, 아티스트, 재생 시간을 화면에 표시합니다. 사용자 친화적인 인터페이스를 제공하기 위해 구분선을 사용하고 명확하게 정보를 정리합니다.

(2) 세부 기능 2. 재생/일시 정지

- 설명: SPACE 키를 입력해 재생과 일시 정지 상태를 토글할 수 있습니다. 일시 정지 상태에서는 메시지를 출력하며, 재생 상태로 전환 시 메시지를 갱신합니다.

(3) 세부 기능 3. 이전/다음 곡 제어

- 설명: 사용자가 N 또는 P 키를 입력하여 다음 곡이나 이전 곡으로 이동할 수 있습니다. 인덱스를 기준으로 순환 처리가 가능하며, 곡 리스트의 끝에 도달하면 처음 또는 마지막 곡으로 이동합니다.

3. 기능 구현

(1) 기능 1. 음악 파일 목록 읽어오기

- 설명: files.h 파일에 구현된 코드를 통해 특정 디렉토리(./songs/)에서 음악 파일 목록을 읽어와 관리합니다. 음악 파일은 .mp3 확장자로 필터링되며, 파일명은 토큰화 과정을 통해 노래 제목, 아티스트, 재생 시간 등으로 분류됩니다. 해당 정보들은 map 구조를 사용해 저장하였습니다.
- 적용된 배운 내용
 - i. 디렉토리 내 파일 탐색: C++의 DIR 및 dirent 구조체를 활용하여 디렉토리 내에서 파일 목록을 읽는 기능 구현
 - ii. 문자열 처리: string 과 토큰화 함수 tokenize() 를 이용해 파일명을 구성 요소로 나누는 작업 진행
 - iii. 데이터 구조: map 을 사용해 인덱스를 기반으로 노래 데이터를 저장하고 관리하는 기능 구현

- 코드 스크린샷

```
void populateSongDetails()
{
    vector<string> files;

    DIR *dir;
    struct dirent *diread;
    if ((dir = opendir("./songs/")) != nullptr)
    {
        while ((diread = readdir(dir)) != nullptr)
        {
            string filename = diread->d_name;
            if (filename.size() > 3 && filename.substr(filename.size() - 3) == "mp3")
            {
                files.push_back(filename);
            }
        }
        closedir(dir);
    }
    else
    {
        perror("opendir");
    }

    for (size_t i = 0; i < files.size(); ++i)
    {
        songList[i + 1] = files[i];

        vector<string> details;
        tokenize(files[i], '-', details);

        if (details.size() >= 3)
        {
            songArtist[i + 1] = details[0];
            songTitle[i + 1] = details[1];
            songDuration[i + 1] = details[2].substr(0, details[2].find('.'));
        }
    }
}
```

(2) 기능 2. 노래 목록 표시 및 선택

- 설명: main.cpp에서 listSongs() 메서드는 사용자가 사용할 수 있는 노래 목록을 화면에 출력합니다. 노래 목록은 populateSongDetails() 함수로 생성된 데이터를 바탕으로 구성되고, 사용자는 해당 목록에서 특정 노래를 선택할 수 있습니다.
- 적용된 배운 내용
 - i. 데이터 표시: map에 저장된 데이터를 반복문으로 읽어 화면에 출력
 - ii. 사용자 입력 처리: 사용자의 선택을 받아 특정 인덱스에 해당하는 노래에 대한 정보 처리
 - iii. 화면 인터페이스: C++의 기본 출력 함수 cout을 활용해 사용자 친화적 인터페이스 구현

■ 코드 스크린샷

```
void listSongs()
{
    clearScreen
    horizontalFill
    cout
    << "*"          사용 가능한 노래 목록          "*" << endl;
    horizontalFill

    populateSongDetails();
    if (songList.empty())
    {
        cout << "현재 './songs/' 디렉토리에 노래가 없습니다." << endl;
    }
    else
    {
        for (const auto &[index, file] : songList)
        {
            cout << index << ". " << getSongTitle(index) << " (" << getSongArtist(index)
            << ") - " << getSongDuration(index) << endl;
        }
    }

    cout << "\n노래를 선택하려면 번호를 입력하세요 (종료: '0')..." << endl;
    int choice;
    cin >> choice;

    if (choice >= 1 && choice <= 9)
    {
        if (songList.find(choice) != songList.end())
        {
            currentSongIndex = choice;
            playSong();
        }
    }
    else if (choice == 0)
    {
        cout << "프로그램을 종료합니다." << endl;
        exit(0);
    }
}
```

(3) 기능 3. 음악 재생 및 정지

- 설명: main.cpp 파일의 playSong() 및 handlePlayback() 메서드는 사용자가 선택한 노래의 정보를 출력하고, 재생/일시 정지 및 메뉴 이동 기능을 제공합니다. 또한 노래 재생은 실제 음원 재생이 아닌 현재 노래 정보를 화면에 표시하는 것으로 구현했으며, 일시 정지 상태에서는 재생 상태가 유지되다가 사용자 입력에 따라 다시 재개됩니다.
- 적용된 배운 내용
 - i. 상태 관리: isPaused 변수로 재생 및 일시 정지 상태 관리
 - ii. 사용자 입력 처리: getchar()를 활용하여 다양한 입력(Space, B, N, P)에 대한 동작 구분
 - iii. 반복 처리 및 이벤트 루프: while 루프를 사용해 재생 상태를 지속적으로 갱신하여 사용자 입력 대기

■ 코드 스크린샷

```
void playSong()
{
    clearScreen
    horizontalFill
    cout
    << "*"           현재 재생 중:           "*" << endl;
    horizontalFill
    cout
    << "제목: " << getSongTitle(currentSongIndex) << endl;
    cout << "아티스트: " << getSongArtist(currentSongIndex) << endl;
    cout << "길이: " << getSongDuration(currentSongIndex) << endl;
    cout << "\n[SPACE] 일시 정지 / 재생    [B] 메뉴로 돌아가기" << endl;
    cout << "[N] 다음 곡    [P] 이전 곡" << endl;

    handlePlayback();
}

void handlePlayback()
{
    while (true)
    {
        input = getchar();

        if (input == ' ')
        {
            isPaused = !isPaused;
            if (isPaused)
            {
                cout << "\n[일시 정지됨] 아무 키나 눌러 계속하세요..." << endl;
            }
            else
            {
                cout << "\n[재생 중] 계속 재생..." << endl;
            }
        }
        else if (input == 'B' || input == 'b')
        {
            isPaused = false;
            return;
        }
        else if (input == 'N' || input == 'n')
        {
            nextSong();
        }
        else if (input == 'P' || input == 'p')
        {
            previousSong();
        }
    }
}
```

(4) 기능 4. 플레이리스트 관리

- 설명: 플레이리스트 관리는 사용자가 선택한 곡을 추가, 삭제, 정렬(셔플), 초기화할 수 있는 기능을 제공합니다. 이 기능은 playlist 벡터와 관련 메서드들 (addToPlaylist, deleteFromPlaylist, clearPlaylist, shufflePlaylist)을 활용하여 구현되었습니다. 사용자는 선택한 곡을 리스트에 추가하거나, 셔플 모드를 통해 랜덤한 순서로 노래를 재생할 수 있습니다.
- 적용된 배운 내용
 - i. **벡터 활용:** vector 를 사용하여 동적으로 플레이리스트를 관리하며, find 및 erase 메서드로 효율적으로 데이터를 삭제함.
 - ii. **셔플링 알고리즘:** std::shuffle과 std::random_device를 활용하여 플레이리스트를 무작위로 정렬하는 셔플 기능 구현.
 - iii. **상태 관리:** isShuffle 변수를 통해 셔플 모드 활성화 여부를 관리.

- 코드 스크린샷

```
void addToPlaylist(int songIndex)
{
    if (find(playlist.begin(), playlist.end(), songIndex) == playlist.end())
    {
        playlist.push_back(songIndex);
    }
}

void deleteFromPlaylist()
{
    clearScreen;
    cout << "*"           Remove from Playlist           "*" << endl;
    if (playlist.empty())
    {
        cout << "The playlist is empty and cannot be removed." << endl;
        return;
    }
    cout << "Enter the number of the song to remove: " << endl;
    int songNumber;
    cin >> songNumber;
    if (songNumber >= 1 && songNumber <= static_cast<int>(playlist.size()))
    {
        playlist.erase(playlist.begin() + (songNumber - 1));
        cout << "The selected song has been removed from the playlist." << endl;
    }
    else
    {
        cout << "Invalid input." << endl;
    }
}

void clearPlaylist()
{
    playlist.clear();
    cout << "The playlist has been cleared." << endl;
}
```

(5) 기능 5. 다음/이전 곡 재생

- 설명: 사용자는 현재 재생 중인 곡에서 다음 곡으로 이동하거나 이전 곡으로 돌아갈 수 있습니다. nextSong 및 previousSong 메서드를 활용하여 셔플 모드와 일반 모드에 따라 올바른 곡을 선택하여 재생합니다.
- 적용된 배운 내용
 - i. **조건부 이동:** find 메서드를 사용하여 현재 곡의 위치를 기준으로 다음 곡 및 이전 곡의 인덱스를 탐색.
 - ii. **순환 처리:** 마지막 곡에서 다시 첫 곡으로 돌아가거나, 첫 곡에서 마지막 곡으로 돌아가는 기능 구현.
 - iii. **셔플 리스트 관리:** 셔플 모드에서는 섞인 리스트(shuffledList)를 기준으로 이동.

- 코드 스크린샷

```
void nextSong()
{
    if (isShuffle)
    {
        currentSongIndex = shuffledList[(find(shuffledList.begin(), shuffledList.end(), currentSongIndex) + 1) % shuffledList.size()];
    }
    else
    {
        auto it = find(playlist.begin(), playlist.end(), currentSongIndex);
        if (it != playlist.end() && ++it != playlist.end())
        {
            currentSongIndex = *it;
        }
        else
        {
            currentSongIndex = playlist[0];
        }
    }

    string nextSongPath = "./songs/" + songList[currentSongIndex];
    playSong(nextSongPath);
}

void previousSong()
{
    if (isShuffle)
    {
        currentSongIndex = shuffledList[(find(shuffledList.begin(), shuffledList.end(), currentSongIndex) - 1) % shuffledList.size()];
    }
    else
    {
        auto it = find(playlist.begin(), playlist.end(), currentSongIndex);
        if (it != playlist.begin())
        {
            currentSongIndex = *(--it);
        }
        else
        {
            currentSongIndex = playlist.back();
        }
    }

    string prevSongPath = "./songs/" + songList[currentSongIndex];
    playSong(prevSongPath);
}
```

(6) 기능 6. 반복 재생 모드

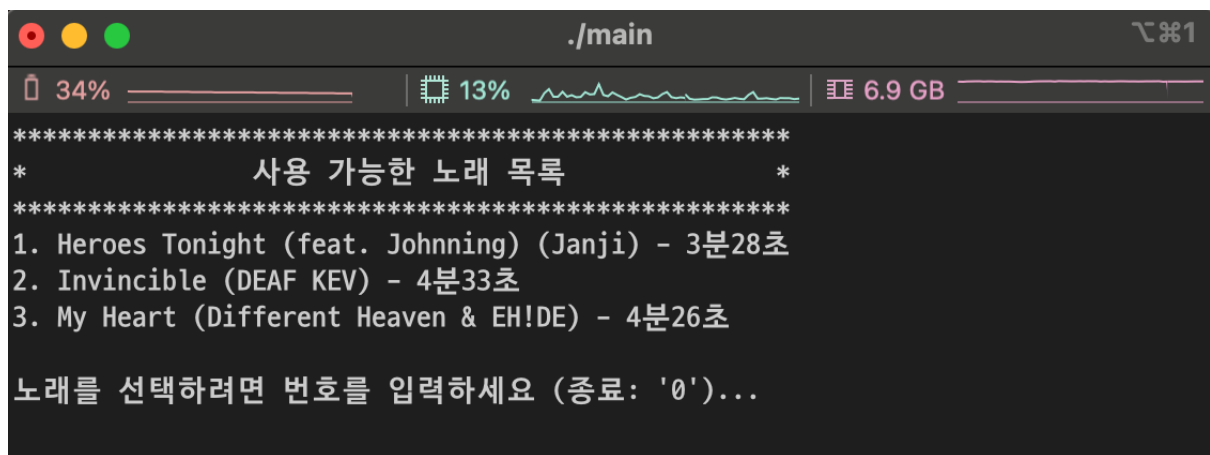
- 설명: toggleRepeat 메서드를 통해 반복 재생 모드를 활성화하거나 비활성화할 수 있습니다. 반복 모드가 활성화된 경우 현재 재생 중인 곡이 종료되면 자동으로 다시 재생됩니다.
- 적용된 배운 내용
 - i. **상태 관리:** isRepeat 변수를 활용하여 반복 재생 모드 활성화 상태를 관리.
 - ii. **사용자 피드백:** 콘솔 출력(cout)으로 모드 상태를 사용자에게 즉시 전달.
- 코드 스크린샷

```
void toggleRepeat()
{
    isRepeat = !isRepeat;
    if (isRepeat)
    {
        cout << "[Repeat mode enabled]" << endl;
    }
    else
    {
        cout << "[Repeat mode disabled]" << endl;
    }
}
```


4. 테스트 결과

(1) 음악 파일 목록 읽어오기

- 설명: files.h 파일에 구현된 코드를 통해 특정 디렉토리(./songs/)에서 음악 파일 목록을 읽어와 관리합니다. 음악 파일은 .mp3 확장자로 필터링되며, 파일명은 토 큰화 과정을 통해 노래 제목, 아티스트, 재생 시간 등으로 분류됩니다. 해당 정보 들은 map 구조를 사용해 저장하였습니다.
- 테스트 결과 스크린샷

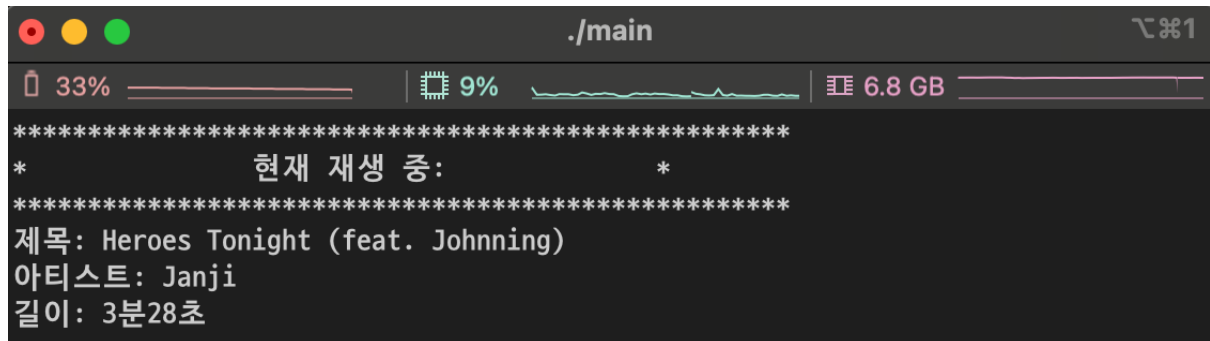


```
./main
34% 13% 6.9 GB
*****
*          사용 가능한 노래 목록          *
*****
1. Heroes Tonight (feat. Johnning) (Janji) - 3분28초
2. Invincible (DEAF KEV) - 4분33초
3. My Heart (Different Heaven & EH!DE) - 4분26초

노래를 선택하려면 번호를 입력하세요 (종료: '0')...
```

(2) 노래 목록 표시 및 선택

- 설명: main.cpp에서 listSongs() 메서드는 사용자가 사용할 수 있는 노래 목록을 화면에 출력합니다. 노래 목록은 populateSongDetails() 함수로 생성된 데이터를 바탕으로 구성되고, 사용자는 해당 목록에서 특정 노래를 선택할 수 있습니다.
- 테스트 결과 스크린샷



```
./main
33% 9% 6.8 GB
*****
*           현재 재생 중:           *
*****
제목: Heroes Tonight (feat. Johnning)
아티스트: Janji
길이: 3분28초
```

(3) 음악 재생 및 정지

- 설명: main.cpp 파일의 playSong() 및 handlePlayback() 메서드는 사용자가 선택한 노래의 정보를 출력하고, 재생/일시 정지 및 메뉴 이동 기능을 제공합니다. 또한 노래 재생은 실제 음원 재생이 아닌 현재 노래 정보를 화면에 표시하는 것으로 구현했으며, 일시 정지 상태에서는 재생 상태가 유지되다가 사용자 입력에 따라 다시 재개됩니다.
- 테스트 결과 스크린샷

```
./main
33% 9% 6.9 GB
*****
*           현재 재생 중:           *
*****
제목: Heroes Tonight (feat. Johnning)
아티스트: Janji
길이: 3분28초

[SPACE] 일시 정지 / 재생    [B] 메뉴로 돌아가기

[일시 정지됨] 아무 키나 눌러 계속하세요...

[재생 중] 계속 재생...
_
```

(4) 플레이리스트 관리

- 설명: 플레이리스트 관리는 사용자가 선택한 곡을 추가, 삭제, 정렬(셔플), 초기화할 수 있는 기능을 제공합니다. 이 기능은 playlist 벡터와 관련 메서드들 (addToPlaylist, deleteFromPlaylist, clearPlaylist, shufflePlaylist)을 활용하여 구현되었습니다. 사용자는 선택한 곡을 리스트에 추가하거나, 셔플 모드를 통해 랜덤한 순서로 노래를 재생할 수 있습니다.
- 테스트 결과 스크린샷

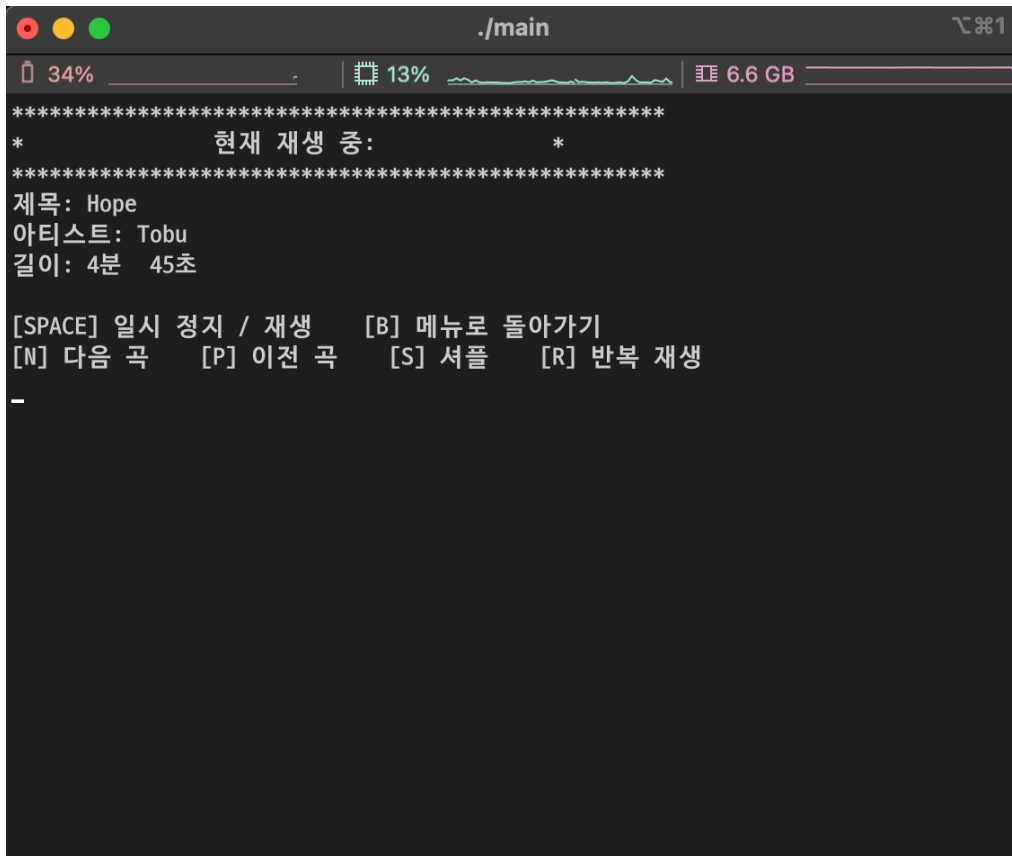
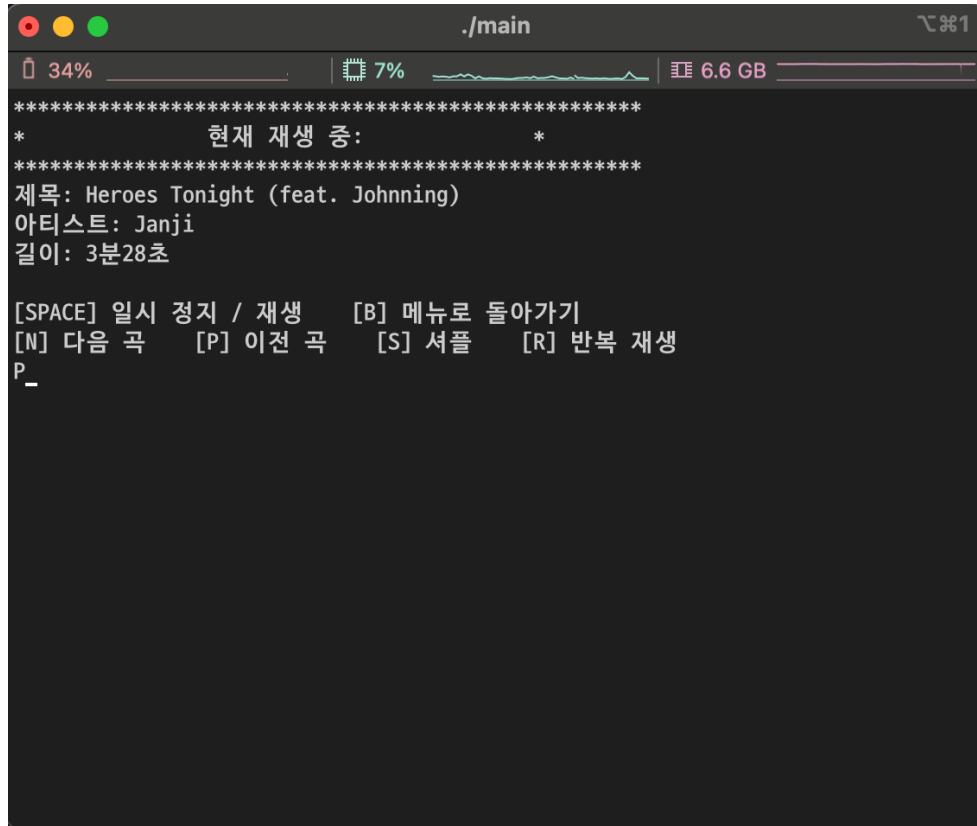
```
./main
34% 10% 6.5 GB
*****
*           현재 재생 중:           *
*****
제목: Heroes Tonight (feat. Johnning)
아티스트: Janji
길이: 3분28초

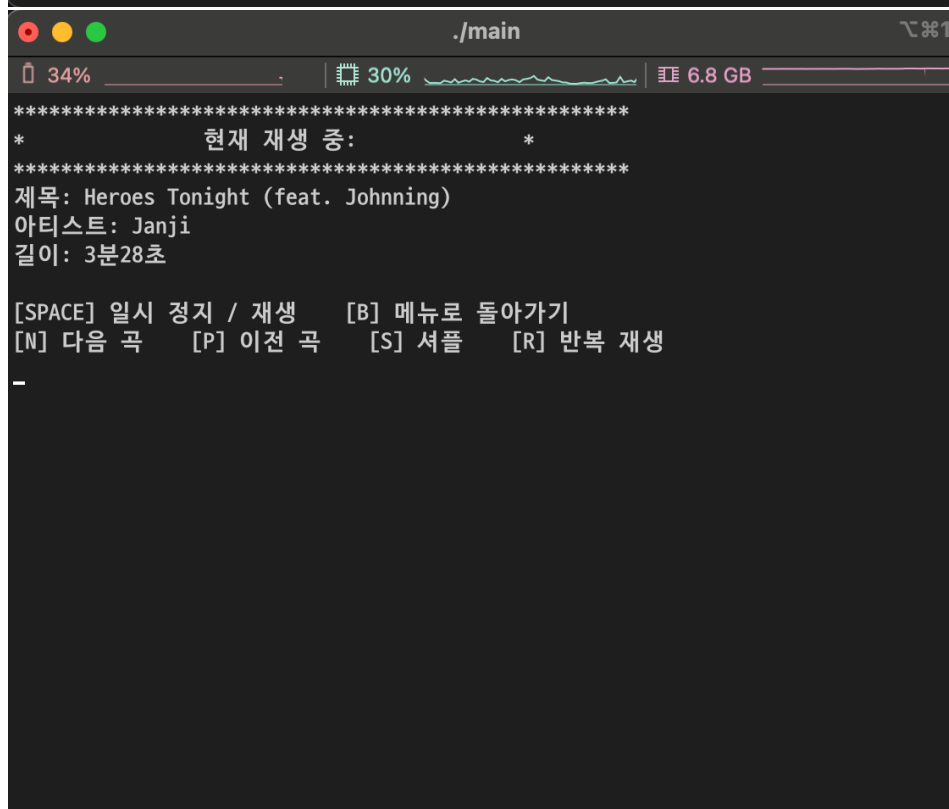
[SPACE] 일시 정지 / 재생   [B] 메뉴로 돌아가기
[N] 다음 곡   [P] 이전 곡   [S] 셔플   [R] 반복 재생
S
[셔플 모드 활성화]
S
[셔플 모드 비활성화]
```

(5) 다음/이전 곡 재생

- 설명: 사용자는 현재 재생 중인 곡에서 다음 곡으로 이동하거나 이전 곡으로 돌아갈 수 있습니다. nextSong 및 previousSong 메서드를 활용하여 셔플 모드와 일반 모드에 따라 올바른 곡을 선택하여 재생합니다.

■ 테스트 결과 스크린샷





(6) 기능 6. 반복 재생 모드

- 설명: toggleRepeat 메서드를 통해 반복 재생 모드를 활성화하거나 비활성화할 수 있습니다. 반복 모드가 활성화된 경우 현재 재생 중인 곡이 종료되면 자동으로 다시 재생됩니다
- 테스트 결과 스크린샷

```
./main
34% 10% 6.5 GB
*****
*           현재 재생 중:           *
*****
제목: Heroes Tonight (feat. Johnning)
아티스트: Janji
길이: 3분28초

[SPACE] 일시 정지 / 재생   [B] 메뉴로 돌아가기
[N] 다음 곡   [P] 이전 곡   [S] 셔플   [R] 반복 재생
S
[셔플 모드 활성화]
S
[셔플 모드 비활성화]
R
[반복 재생 활성화]
R
[반복 재생 비활성화]
-
```


5. 계획 대비 변경 사항

5-1. 기존의 기능 계획

[184128 박지환] 1차_프로젝트_진척보고서.docx

1) 기능 1. 음악 추가 및 삭제

- 설명: 사용자가 원하는 음악 파일을 프로그램에 추가하거나 삭제하는 기능입니다.

(1) 세부 기능 1. 음악 파일 추가

- 설명: 사용자가 음악 파일의 경로를 입력하여 플레이리스트에 추가하는 기능입니다.

(2) 세부 기능 2. 음악 파일 삭제

- 설명: 사용자가 플레이리스트에서 특정 음악 파일을 삭제할 수 있는 기능입니다.

(3) 세부 기능 3. 음원 유효성 검사

- 설명: 추가한 유튜브 링크가 올바른 링크인지 확인하여, 잘못된 파일 추가를 방지합니다.

2) 기능 2. 음악 재생 및 정지

- 설명: 사용자가 선택한 음악을 재생하고 정지할 수 있는 기능입니다.

(1) 세부 기능 1. 선택한 음악 재생

- 설명: 플레이리스트에서 특정 음악을 선택하여 재생하는 기능입니다.

(2) 세부 기능 2. 재생 중 음악 정지

- 설명: 현재 재생 중인 음악을 일시정지하거나 정지할 수 있는 기능입니다.

(3) 세부 기능 3. 음악 재개

- 설명: 일시 정지된 음악을 재개하여 음악 감상의 흐름을 유지하도록 하는 기능입니다.

3) 기능 3. 플레이리스트 관리

- 설명: 사용자가 자신만의 플레이리스트를 관리할 수 있는 기능입니다.

(1) 세부 기능 1. 새로운 플레이리스트 생성

- 설명: 사용자가 새로운 플레이리스트를 만들고 파일을 추가할 수 있는 기능입니다.

(2) 세부 기능 2. 플레이리스트 삭제

- 설명: 특정 플레이리스트를 삭제하여 관리 효율성을 높이도록 하는 기능입니다.

(3) 세부 기능 3. 플레이리스트 내 음악 정렬

- 설명: 추가된 음악을 이름, 가수명, 추가된 날짜에 따라 정렬하는 기능입니다.

4) 기능 4. 재생 모드 설정

- 설명: 다양한 재생 모드를 제공하여 사용자가 선호하는 방식으로 음악을 감상할 수 있는 기능입니다.

(1) 세부 기능 1. 랜덤 재생

- 설명: 플레이리스트 내 음악을 무작위 순서로 재생하는 기능입니다.

(2) 세부 기능 2. 반복 재생

- 설명: 현재 재생 중인 음악을 반복해서 재생하는 기능입니다.

(3) 세부 기능 3. 플레이리스트 전체 반복

- 설명: 플레이리스트의 모든 곡을 반복 재생하는 기능입니다.

5) 기능 5. 플레이리스트 저장 및 복원

- 설명: 프로그램 종료 시 추가해둔 플레이리스트가 저장되며, 이후 프로그램 재

실행 시 동일한 플레이리스트로 복원되는 기능입니다.

(1) 세부 기능 1. 플레이리스트 파일 저장

- 설명: 사용자가 추가한 유튜브 URIL이나 음악 정보가 텍스트 파일로 저장되어, 프로그램 종료 후에도 유지될 수 있도록 하는 기능입니다.

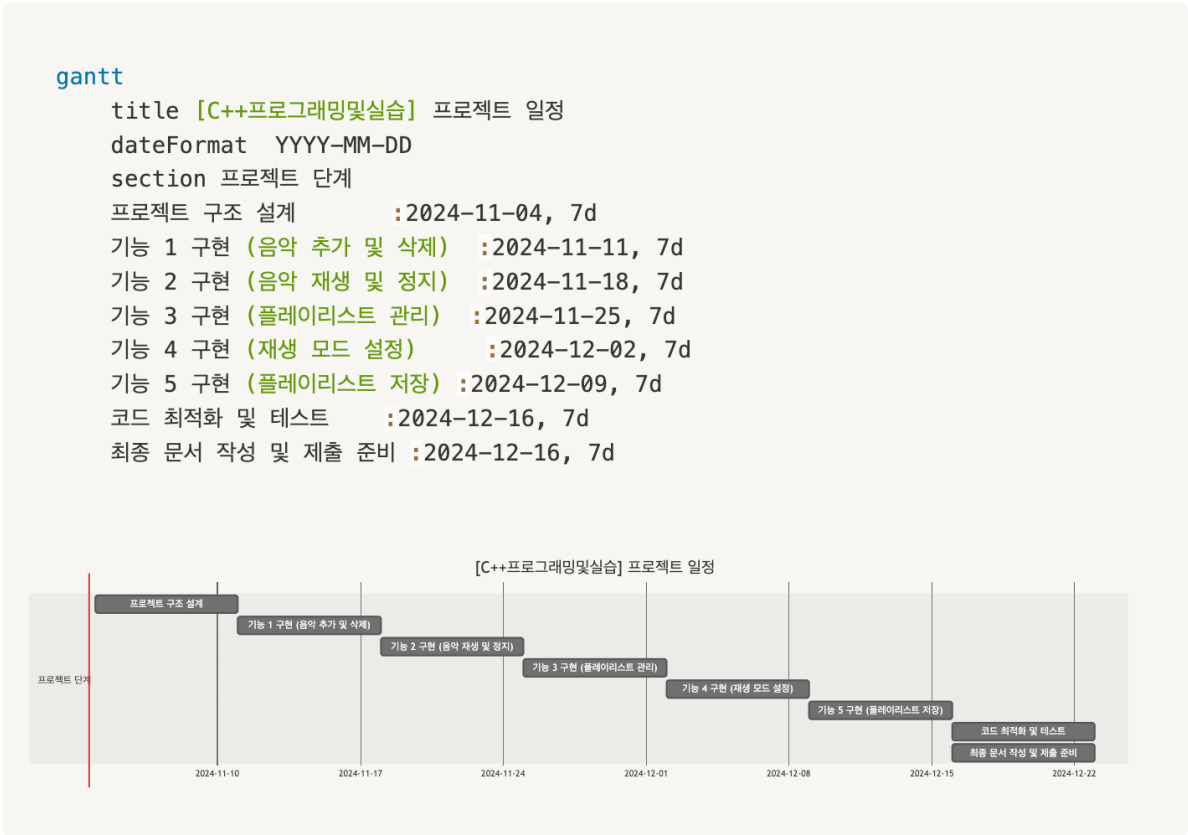
(2) 세부 기능 2. 플레이리스트 파일 불러오기

- 설명: 프로그램 시작 시 저장된 플레이리스트 파일을 불러와 이전 상태를 그대로 복원할 수 있도록 하는 기능입니다.

5-2. 변경된 기능 계획

[184128 박지환] 2차_프로젝트_진척보고서.docx / 2. 기능 계획

5-3. 기존의 프로젝트 일정



날짜	진행 내용
11월 4일 ~ 11월 10일	프로젝트 구조 설계
11월 11일 ~ 11월 17일	기능 1 구현 (음악 추가 및 삭제)
11월 18일 ~ 11월 24일	기능 2 구현 (음악 재생 및 정지)
11월 25일 ~ 12월 1일	기능 3 구현 (플레이리스트 관리)
12월 2일 ~ 12월 8일	기능 4 구현 (재생 모드 설정)
12월 9일 ~ 12월 15일	기능 5 구현 (플레이리스트 저장 및 복원)
12월 16일 ~ 12월 21일	코드 최적화 및 테스트, 최종 문서 작성 및 프로젝트 제출 준비

5-4. 변경된 프로젝트 일정

[184128 박지환] 2차_프로젝트_진척보고서.docx / 5. 프로젝트 일정

5-5. 기존의 개발 환경

- 설명: 기존 MacOS에서 작업 후 Windows 환경에서 호환이 되는지 확인한 결과,
코드가 전혀 실행되지 않았음

5-6. 변경된 개발 환경

- 설명: Windows 환경에서 코드가 실행되도록 모든 코드를 수정하는 작업 시행

```

1  #ifndef DEFINE_LIBRARY
2  #define DEFINE_LIBRARY
3
4  #include <iostream>
5  #include <cstdlib>
6  #ifdef _WIN32
7  #include <windows.h>
8  #else
9  #include <unistd.h>
10 #endif
11
12 using namespace std;
13
14 #ifdef _WIN32
15 #define clearScreen system("cls")
16 inline void gotoxy(int x, int y)
17 {
18     COORD coord;
19     coord.X = x;
20     coord.Y = y;
21     SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
22 }
23 #else
24 #define clearScreen system("clear")
25 #define gotoxy(x, y) printf("\033[%d;%dH", (y), (x))
26 #endif
27
28 #define horizontalFill cout << "*****\n";
29 #define horizontalBlank cout << " " * \n";
30
31 #endif

```

6. 느낀점

6-1. 어려웠던 점 및 해결 방안 1

프로젝트를 진행하면서 다양한 라이브러리(ffmpeg 또는 ffplay, PortAudio, SDL2 등)를 사용하여 음원을 재생하려 했습니다. 그러나 이러한 라이브러리를 사용하는 동안 프로그램의 다른 기능을 실행할 수 없는 문제가 발생했습니다.

이를 해결하기 위해 다양한 방법을 모색했으나, 다른 해결책을 찾지 못하였고 결과적으로 라이브러리를 사용하여 노래가 재생되는 동안은 프로그램 내에 있는 다음 곡/이전 곡, 셔플 등의 기능 사용에 제한이 생겼습니다.

이처럼 라이브러리 사용의 어려움을 겪으면서, 프로젝트를 진행하는 데 필요한 기능을 효과적으로 구현하기 위해서는 무겁고 복잡한 라이브러리보다는 CLI 환경에 최적화된 가벼운 코드만을 사용하는 것이 중요하다는 것을 알았고, 또한 필요한 기능을 구현하는 데 필수적인 코드들만 사용해 리소스 효율성을 높이고 사용자 요구사항에 맞게 커스터마이징하는 것이 중요하다는 것도 깨달았습니다.

6-2. 어려웠던 점 및 해결 방안 2

그리고 프로젝트를 진행하면서 또 다른 어려움은 플랫폼 간 호환성 문제였습니다.

초기에는 MacOS 환경에서 개발을 진행하였지만, 실습실의 환경이나 다른 학생분들의 개발 환경도 고려해보니 Windows에서도 실행되는 프로그램으로 개발해야겠다는 생각이 문득 들게되어 Windows 환경에서 실행해 보니 코드가 전혀 실행되지 않는 문제가 발생했습니다.

이를 해결하기 위해 프로젝트 일정을 조정하고 Windows 환경에서 코드가 실행되도록 모든 코드를 수정하는 작업을 진행하였고, 이렇게 호환성 문제를 해결하기 위해 추가적인 개발 시간이 필요했고 전체적인 일정이 조금 바뀌게 되었습니다.

결과적으로는 다양한 운영체제에서 프로그램을 실행할 수 있는 코드를 얻을 수 있었습니다.

이 과정에서 개발 시 플랫폼 간 호환성을 고려하는 것이 매우 중요하다는 교훈을 얻을

수 있었습니다.

특정 운영체제에 최적화된 코드를 작성하다 보면 다른 환경에서는 문제가 발생할 수 있기 때문에, 초기 개발 단계부터 다양한 플랫폼에서 실행을 검증하고 호환성을 확보하는 것이 필요하다고 생각이 들었고, 향후에 다른 프로젝트를 함께 있어서도 프로젝트 계획 수립 시 이러한 점을 고려하여 충분한 개발 시간을 확보할 계획입니다.

6-3. 결론

프로젝트를 진행하면서 다양한 기술적 도전과 문제를 해결하는 경험을 할 수 있었습니다.

특히, 음원 재생을 위한 라이브러리 사용의 어려움과 플랫폼 간 호환성 문제를 해결하는 과정에서 많은 것을 배울 수 있었습니다.

라이브러리 사용 시 프로그램의 다른 기능을 실행할 수 없는 문제를 겪었지만, 이러한 과정에서 해결책을 찾기 위해 제한된 자원 내에서 유연하게 접근하는 방법을 모색했습니다.

또한 MacOS에서 개발하던 프로그램이 Windows에서 원활하게 실행되지 않는 문제를 겪었지만, 이를 해결하기 위해 다양한 플랫폼에서의 실행을 검증하고 호환성을 확보하는 중요성 또한 깨달았습니다.

이러한 경험을 통해 향후에는 더욱 완성도 높은 프로그램을 개발해볼 계획이며, 이번 프로젝트를 통해 얻은 경험은 앞으로의 개발 과정에 큰 도움을 줄 것이라고 생각하며 이번 보고서를 마치겠습니다.