**2018-2 자료구조**

**프로젝트 2 명세서**

**학과 : 컴퓨터공학과**

**학번 : 2015104214**

**이름 : 장유진**

|  |
| --- |
| 1. 프로젝트 목적 |

* 시중에 나와 있는 곰플레이어나 KM 플레이어 같은 음원 관리 프로그램들처럼 음원을 재생하고 음원, 앨범, 재생목록 등을 관리하는 프로그램을 만든다. 프로젝트1에서 만든 프로그램을 분석하고 수업시간에 배운 다양한 자료구조를 사용하여 더 보완된 프로그램을 만드는 것이 목표이다.

|  |
| --- |
| 1. 프로그램 기능 |

* 1. 음원들을 재생하기 위한 곡 목록 추가, 제거, 갱신 등의 관리와 정렬이 주된 기능이다.
  2. 음원은 ‘곡명, 가수명, 작곡자, 작사자, 앨범명, 장르, 가사’를 레코드로 가진다.
  3. 앨범은 ‘앨범명, 가수명, 수록곡리스트, 수록곡 수’를 레코드로 가지며 수록곡은 간단한 음원정보를 리스트로 한다.
  4. 간단한 음원정보는 ‘곡명, 가수명, 재생횟수’를 레코드로 가진다.
  5. 아티스트는 ‘가수명, 발매한 곡리스트’를 레코드로 가지며 발매한 곡리스트는 간단한 음원정보를 리스트로 한다.
  6. 전체 노래 목록(Song List)은 Binary Search Tree의 구조로 정렬되고, 곡명이 Primary Key가 된다.
  7. 앨범 목록(Album List)은 Binary Search Tree의 구조로 정렬되고, 앨범명이primary key가 된다. 각 앨범에는 간단한 음원정보로 된 수록곡 리스트가 있다.
  8. 가수 목록(Artist List)은 Binary Search Tree의 구조로 정렬되고, 가수명이 primary key가 된다. 각 가수에는 간단한 음원정보로 된 발매한 곡리스트가 있다.
  9. 장르 목록(Genre List)은 Binary Search Tree의 구조로 정렬되고, 장르명이 primary key가 된다. 각 장르에는 간단한 음원정보로 된 장르에 속한 곡리스트가 있다.
  10. 유저는 최대100곡까지 들어가는 간단한 음원정보로 된 배열의 재생 목록이 있다. 이곳에 곡을 추가하여 노래를 재생한다.
  11. 유저의 재생을 바탕으로 가장 많이 재생한 곡 30개를 저장하는 간단한 음원정보로 된 배열의 목록이 있다.
  12. 음원은 ‘곡명, 가수명, 앨범명, 장르’로 검색 가능하다.
  13. 프로그램 실행 중 파일 입출력으로 DB를 받아오거나 파일로 저장할 수 있다.

|  |
| --- |
| 1. 클래스 목록 |

|  |  |
| --- | --- |
| 클래스명 | 기능 |
| AlbumType | 앨범의 정보가 저장될 클래스이다. 가수명, 앨범명, 수록곡 리스트, 수록곡 수가 멤버변수로 선언된다. |
| Application | 프로그램의 기능을 총괄한다. 프로그램이 실행되는 동안 사용자가 선택할 수 있는 메뉴를 출력하고 실행한다. 음원의 재생 및 음원목록과 재생목록을 관리하고, 파일 입출력을 수행한다. |
| ArtistType | 가수의 정보가 저장될 클래스이다. 가수명, 발매한 곡리스트가 멤버변수로 선언된다. |
| BinarySearchTree | Template 구조를 가진 Binary Search Tree 클래스이다. 이 클래스를 이용하여 뮤직, 앨범, 장르, 가수 리스트를 만든다. |
| GenreType | 장르의 정보가 저장될 클래스이다. 장르명, 장르에 속한 곡리스트, 곡 수가 멤버변수로 선언된다. |
| MusicType | 음원의 정보가 저장될 클래스이다. 곡명, 가수명, 작곡가, 작사가, 앨범명, 장르, 가사가 멤버변수로 선언된다. |
| SimpleMusicType | 간단한 음원의 정보가 저장될 클래스이다. 곡명, 가수명, 재생횟수가 멤버변수로 선언된다. |

|  |
| --- |
| 1. 시나리오 흐름도(Diagram) |
| 1. 헤더 파일 |
| AlbumType.h  class AlbumType  {  public:  AlbumType(); // 디폴트 생성자  ~AlbumType(); // 소멸자  /\*\*  \* @brief 앨범에 수록곡 추가  \* @pre 앨범이 생성되어있어야함  \* @post 앨범에 수록곡 추가됨  \* @param inData 추가할 곡의 정보  \*/  void Add(SimpleMusicType inData); // 수록곡 추가    /\*\*  \* @brief 앨범에서 수록곡 삭제  \* @pre 앨범이 생성되어있어야함  \* @post 앨범에서 수록곡 삭제됨  \* @param inData 삭제할 곡의 정보  \*/  void Delete(SimpleMusicType inData); // 수록곡 삭제  /\*\*  \* @brief 가수명 가져오기  \* @pre 가수명이 설정되있어야함  \* @return 가수명  \*/  string GetArtistName(); // 가수명 가져오기  /\*\*  \* @brief 앨범명 가져오기  \* @pre 앨범명이 설정되있어야함  \* @return 앨범명  \*/  string GetAlbumName(); // 앨범명 가져오기    /\*\*  \* @brief 수록곡 수 가져오기  \* @pre 수록곡 수가 설정되있어야함  \* @return 수록곡 수  \*/  int GetNumberOfContainMusic(); // 수록곡 수 가져오기  /\*\*  \* @brief 가수명 설정하기  \* @post 가수명 설정  \* @param inArtistName 설정할 가수명  \*/  void SetArtistName(string inArtistName); // 가수명 설정  /\*\*  \* @brief 앨범명 설정하기  \* @post 앨범명 설정  \* @param inAlbumName 설정할 앨범명  \*/  void SetAlbumName(string inAlbumName);// 앨범명 설정    /\*\*  \* @brief 모든 레코드 설정하기  \* @post 모든 레코드 설정  \* @param inArtistName 설정할 가수명  \* @param inAlbumName 설정할 앨범명  \*/  void SetRecord(string inArtistName, string inAlbumName); // 모든 레코드 설정  /\*\*  \* @brief 화면에 가수명 출력  \* @pre 가수명이 설정되어있어야함  \*/  void DisplayArtistNameOnScreen(); // 화면에 가수명 출력  /\*\*  \* @brief 화면에 앨범명 출력  \* @pre 앨범명이 설정되어있어야함  \*/  void DisplayAlbumNameOnScreen(); // 화면에 앨범명 출력    /\*\*  \* @brief 화면에 수록곡 수 출력  \* @pre 수록곡 수가 설정되어있어야함  \*/  void DisplayNumberOfContainMusicOnScreen(); // 화면에 수록곡 수 출력  /\*\*  \* @brief 화면에 수록곡 리스트 출력  \* @pre 수록곡 리스트가 설정되어있어야함  \*/  void DisplayContainMusicListOnScreen();// 화면에 수록곡리스트 출력  /\*\*  \* @brief 화면에 모든 레코드 출력  \* @pre 모든 레코드가 설정되어있어야함  \*/  void DisplayRecordOnScreen(); // 화면에 모든 레코드 출력  // 연산자 오버로딩  /\*\*  \* @brief 앨범클래스 간에 == 연산자 오버로딩  \* @param obj 해당 클래스와 연산될 피연산자 클래스  \*/  bool operator== (AlbumType &obj)  {  if (AlbumName == obj.GetAlbumName())  return true;  else  return false;  }  /\*\*  \* @brief 앨범클래스 간에 > 연산자 오버로딩  \* @param obj 해당 클래스와 연산될 피연산자 클래스  \*/  bool operator> (AlbumType &obj)  {  if (AlbumName > obj.GetAlbumName())  return true;  else  return false;  }  /\*\*  \* @brief 앨범클래스 간에 < 연산자 오버로딩  \* @param obj 해당 클래스와 연산될 피연산자 클래스  \*/  bool operator< (AlbumType &obj)  {  if (AlbumName < obj.GetAlbumName())  return true;  else  return false;  }  /\*\*  \* @brief 앨범클래스의 ostream시 << 연산자 오버로딩  \* @param out ostream시 할 out 종류  \* @param album out과 연산될 피연산자 앨범클래스  \*/  friend void operator<<(ostream& out, AlbumType& album);  private:  string AlbumName; ///< 앨범명(primary key)  string ArtistName; ///< 가수명  BinarySearchTree<SimpleMusicType> m\_List; ///< 수록곡리스트  int m\_Length; ///< 수록곡 수  };  Application.h  class Application  {  public:  /\*\*  \* 디폴트 생성자  \*/  Application()  {  m\_Command = 0;  }  /\*\*  \* 소멸자  \*/  ~Application() {}  /\*\*  \* @brief 프로그램 드라이버  \* @pre 프로그램이 실행되어 있어야 됨.  \* @post 프로그램이 종료됨.  \*/  void Run();  /\*\*  \* @brief 관리자계정 인터페이스  \* @pre Run함수가 실행되어 있어야 됨.  \* @post 계정선택 창으로 돌아감  \*/  void MasterMode();  /\*\*  \* @brief 유저계정 드라이버  \* @pre Run함수가 실행되어 있어야 됨.  \* @post 계정선택 창으로 돌아감  \*/  void UserMode();  /\*\*  \* @brief 리스트에 새로운 Music 추가  \* @pre 리스트가 초기화 되어있어야 됨  \* @post 리스트에 새로운 Music 추가  \* @return 작동이 잘 되었으면 1, 안되었으면 0 리턴  \*/  int AddMusic();  /\*\*  \* @brief 리스트에서 Music 하나 삭제  \* @pre 리스트가 초기화 되어있어야 됨  \* @post 리스트에 일치하는 Music이 있으면 삭제  \* @return 삭제 했으면 1, 없어서 못했으면 0 리턴  \*/  int DeleteMusic();  /\*\*  \* @brief 리스트에서 Music 하나의 데이터 갱신  \* @pre 리스트가 초기화 되어있어야 됨  \* @post 리스트에 일치하는 Music이 있으면 데이터 갱신  \* @return 갱신 했으면 1, 없어서 못했으면 0 리턴  \*/  int ReplaceMusic();  /\*\*  \* @brief 재생목록에서 Music 하나 재생  \* @pre 재생목록이 초기화 되어있어야 됨  \* @post 음악이 재생되고 곡마다 재생된 횟수만큼 재생횟수가 증가  \* @param inNumber 재생할 곡의 리스트 내 순서  \*/  void playMusic(int inNumber);  /\*\*  \* @brief 재생목록에 Music 하나 추가  \* @param inSong 재생목록에 추가할 곡의 곡명  \* @return 추가 했으면 1, 못했으면 0 리턴  \*/  int AddMusicToPlayList(string inSong);  /\*\*  \* @brief 재생목록에서 Music 하나 삭제  \* @return 삭제 했으면 1, 없어서 못했으면 0 리턴  \*/  int deleteMusicFromPlayList();  /\*\*  \* @brief 재생목록에서 Music 찾기  \* @pre 재생목록이 초기화 되어있어야 됨  \* @param inData 재생목록에서 찾는 곡의 곡명  \* @return 찾으면 해당 곡의 순서, 없어서 못찾았으면 -1 리턴  \*/  int GetSeqNumOfPlayList(SimpleMusicType inData);  /\*\*  \* @brief 리스트에서 곡명이 일치하는 모든 Music 정보 출력  \* @pre none  \* @return 출력되었으면 1, 출력안되면 0 리턴  \*/  int SearchByName();  /\*\*  \* @brief 리스트에서 가수명이 일치하는 모든 Music 정보 출력  \* @pre none  \* @return 출력되었으면 1, 출력안되면 0 리턴  \*/  int SearchByArtist();  /\*\*  \* @brief 리스트에서 앨범이 일치하는 모든 앨범 정보 출력  \* @pre none  \* @return 출력되었으면 1, 출력안되면 0 리턴  \*/  int SearchByAlbum();  /\*\*  \* @brief 리스트에서 장르가 일치하는 모든 Music 정보 출력  \* @pre none  \* @return 출력되었으면 1, 출력안되면 0 리턴  \*/  int SearchByGenre();  /\*\*  \* @brief 재생목록 출력  \* @pre none.  \* @post none.  \*/  void DisplayPlayList();  /\*\*  \* @brief 모든 Music 출력  \* @pre none.  \* @post none.  \*/  void DisplayAllMusic();  /\*\*  \* @brief 가장 많이 재생된 30곡 출력  \* @pre none.  \* @post none.  \*/  void DisplayMostPlayList();  /\*\*  \* @brief 리스트 내에 모든 Music 삭제  \* @pre 리스트가 초기화 되어있어야 됨  \* @post 리스트 내에 모든 Music 삭제  \* @return 작동 되었으면 1, 작동 안되었으면 0 리턴  \*/  int MakeEmpty();  /\*\*  \* @brief 파일 설명자별 파일을 입력 파일로 열기  \* @pre 읽기 위한 파일이 존재해야 함  \* @post 읽기 위한 파일 open  \* @param fileName 읽기 위한 파일의 이름  \* @return 작동 되었으면 1, 작동 안되었으면 0 리턴  \*/  int OpenInFile(char \*fileName);  /\*\*  \* @brief 파일 설명자별 파일을 출력 파일로 열기  \* @pre 리스트가 초기화 되어 있어야 함  \* @post 쓰기 위한 파일 open  \* @param fileName 쓰기 위한 파일의 이름  \* @return 작동 되었으면 1, 작동 안되었으면 0 리턴  \*/  int OpenOutFile(char \*fileName);  /\*\*  \* @brief 읽기 모드로 파일 Open, 파일에서 모든 데이터 읽기, 데이터로부터 리스트 설정  \* @pre 파일이 열려있지 않아야 함  \* @post 리스트가 파일로 부터 모든 레코드를 갖고 있어야 함  \* @return 작동 되었으면 1, 작동 안되었으면 0 리턴  \*/  int ReadDataFromFile();  /\*\*  \* @brief 쓰기 모드로 파일 Open, 모든 데이터를 파일에 쓰기  \* @pre 파일이 열려있지 않아야 함  \* @post 파일에 리스트 데이터 저장  \* @return 작동 되었으면 1, 작동 안되었으면 0 리턴  \*/  int WriteDataToFile();  private:  int m\_Command; ///< current command number.  ifstream m\_InFile; ///< input file descriptor.  ofstream m\_OutFile; ///< output file descriptor.  BinarySearchTree<MusicType> m\_SongList; ///< Master list  BinarySearchTree<GenreType> m\_GenreList; ///< Genre list.  BinarySearchTree<ArtistType> m\_ArtistList; ///< Artist list.  BinarySearchTree<AlbumType> m\_AlbumList; ///< Album list.  SimpleMusicType\* m\_MostPlayList = new SimpleMusicType[30]; ///< 가장 많이 재생한 곡 30개  SimpleMusicType\* m\_PlayList = new SimpleMusicType[100]; ///< 재생 목록  int m\_MostPlayListLength; ///< 가장 많이 재생한 곡 목록 길이  int m\_PlayListLength; ///< 재생목록 길이  };  ArtistType.h  class ArtistType  {  public:  ArtistType(); // 디폴트 생성자  ~ArtistType(); // 소멸자  /\*\*  \* @brief 아티스트에 노래 추가  \* @pre 아티스트가 생성되어있어야함  \* @post 아티스트에 노래 추가됨  \* @param inData 추가할 노래의 정보  \*/  void Add(SimpleMusicType inData); // 노래 추가  /\*\*  \* @brief 아티스트에서 노래 삭제  \* @pre 아티스트가 생성되어있어야함  \* @post 아티스트에서 노래 삭제됨  \* @param inData 삭제할 노래의 정보  \*/  void Delete(SimpleMusicType inData); // 노래 삭제    /\*\*  \* @brief 가수명 가져오기  \* @pre 가수명이 설정되있어야함  \* @return 가수명  \*/  string GetArtistName(); // 가수명 가져오기  /\*\*  \* @brief 앨범 개수 가져오기  \* @pre 앨범 개수가 설정되있어야함  \* @return 앨범 개수  \*/  int GetLength(); // 앨범 개수 가져오기    /\*\*  \* @brief 가수명 설정하기  \* @post 가수명 설정  \* @param inArtistName 설정할 가수명  \*/  void SetArtistName(string inArtistName); // 가수명 설정  /\*\*  \* @brief 가수명 설정하기  \* @post 가수명 설정  \* @param inArtistName 설정할 가수명  \*/  void SetRecord(string inArtistName); // 가수명과 생년월일 설정  /\*\*  \* @brief 화면에 가수명 출력  \* @pre 가수명이 설정되어있어야함  \*/  void DisplayArtistNameOnScreen(); // 화면에 가수명 출력  /\*\*  \* @brief 화면에 곡리스트 출력  \* @pre 곡리스트가 설정되어있어야함  \*/  void DisplaySongListOnScreen(); // 화면에 곡리스트 출력  // 연산자 오버로딩  /\*\*  \* @brief 아티스트클래스 간에 == 연산자 오버로딩  \* @param obj 해당 클래스와 연산될 피연산자 클래스  \*/  bool operator== (ArtistType &obj)  {  if (ArtistName.compare(obj.GetArtistName()) == 0)  return true;  else  return false;  }  /\*\*  \* @brief 아티스트클래스 간에 > 연산자 오버로딩  \* @param obj 해당 클래스와 연산될 피연산자 클래스  \*/  bool operator> (ArtistType &obj)  {  if (ArtistName.compare(obj.GetArtistName()) == 1)  return true;  else  return false;  }  /\*\*  \* @brief 아티스트클래스 간에 < 연산자 오버로딩  \* @param obj 해당 클래스와 연산될 피연산자 클래스  \*/  bool operator< (ArtistType &obj)  {  if (ArtistName.compare(obj.GetArtistName()) == -1)  return true;  else  return false;  }  private:  string ArtistName; ///< 가수명(primary key)  BinarySearchTree<SimpleMusicType> m\_List; ///< 노래목록  int m\_Length; ///< 앨범 개수  };  BinarySearchTree.h  /\*\*  \* @brief Tree를 구성하는 Node (node data, left 포인터, right 포인터)  \*/  template<typename T>  struct Node  {  T data; ///< 현 노드의 데이터.  Node\* left; ///< 현 노드보다 작은 값을 가진 노드  Node\* right; ///< 현 노드보다 큰 값을 가진 노드  };  /\*\*  \* @brief Binary Search Tree  \*/  template<typename T>  class BinarySearchTree  {  public:  // 생성자  BinarySearchTree();  // 소멸자  ~BinarySearchTree() { }  /\*\*  \* @brief Tree가 Empty인지 확인  \* @pre none  \* @post none  \* @return Tree가 비어있으면 true리턴, 아니면 false 리턴  \*/  bool IsEmpty()const;  /\*\*  \* @brief Tree가 Full인지 확인  \* @pre none  \* @post none  \* @return Tree가 Full이면 true 리턴, 아니면 false 리턴  \*/  bool IsFull()const;  /\*\*  \* @brief Tree를 비운다  \* @pre none  \* @post none  \*/  void MakeEmpty();  /\*\*  \* @brief Tree에 node가 몇 개인지 확인함  \* @pre none  \* @post none  \* @return Tree의 node 개수를 리턴  \*/  int GetLength() const;  /\*\*  \* @brief Tree에 새 node를 추가한다  \* @pre T의 객체를 생성  \* @post Tree에 새 node가 추가됨  \* @param item 추가할 객체  \*/  void Add(T item);  /\*\*  \* @brief Tree에서 지우고자 하는 node를 찾고 지움  \* @pre T의 객체를 생성  \* @post Tree에서 입력한 node가 삭제됨  \* @param item 삭제할 객체  \*/  void DeleteItem(T item);  /\*\*  \* @brief 리스트에서 객체 갱신  \* @pre 리스트가 설정되어있어야함  \* @post 리스트에서 객체 갱신됨  \* @param item 갱신할 객체  \*/  void ReplaceItem(T& item); // 객체 갱신  /\*\*  \* @brief 입력한 값의 node를 Tree에서 검색함  \* @pre 찾고자 하는 item과 검색결과에 대한 found 설정  \* @post node가 Tree에 있는지 검색결과를 알려줌  \* @param item 찾고자 하는 객체  \* @param found 찾았으면 true, 못찾았으면 false 입력  \*/  void RetrieveItem(T& item, bool &found)const;  /\*\*  \* @brief Tree의 node를 스크린에 출력한다.  \* @pre none  \* @post 스크린에 Tree가 InOrder, PreOrder, PostOrder 방법으로 각각 출력됨.  \* @param out 출력 방식  \*/  void PrintTree(ostream &out)const;  /\*\*  \* @brief Tree의 node를 파일에 입력한다.  \* @pre none  \* @post 스크린에 Tree가 InOrder, PreOrder, PostOrder 방법으로 각각 출력됨.  \* @param out 출력 방식  \*/  void WriteTree(ofstream &out)const;  /\*\*  \* @brief 바이너리서치트리 간에 딥카피를 위한 = 연산자 오버로딩  \* @param tree 해당 클래스와 연산될 피연산자 클래스  \*/  BinarySearchTree<T>& operator=(const BinarySearchTree<T>& tree)  {  MakeEmpty();  CopyNode(root, tree.root);  return \*this;  }  private:  Node<T>\* root; // Node 타입의 root  };  GenreType.h  class GenreType  {  public:  GenreType();  ~GenreType();  /\*\*  \* @brief 장르 가져오기  \* @pre 장르가 설정되있어야 함  \* @post none.  \* @return 장르  \*/  string GetGenre()  {  return genre;  }  /\*\*  \* @brief 장르 설정  \* @pre none.  \* @post 장르 설정  \* @param inGenre 장르  \*/  void SetGenre(string inGenre)  {  genre = inGenre;  }  /\*\*  \* @brief 화면에 장르 출력  \* @pre 장르가 설정되있어야 함  \* @post none  \*/  void DisplayGenreOnScreen()  {  cout << "\t장르 : " << genre << endl;  };  /\*\*  \* @brief 화면에 장르리스트 출력  \* @pre 장르리스트가 설정되있어야 함  \* @post none  \*/  void DisplayListOnScreen();  /\*\*  \* @brief 현재 리스트의 아이템 갯수 가져오기  \* @pre none.  \* @post none.  \* @return 현재 리스트의 아이템 갯수  \*/  int GetLength();  /\*\*  \* @brief 장르리스트에 새로운 Music 추가  \* @pre 장르리스트가 초기화 되어있어야 됨  \* @post 장르리스트에 새로운 Music 추가됨  \* @param inData 추가할 새로운 Music  \*/  void Add(SimpleMusicType inData);  /\*\*  \* @brief 장르리스트에서 Music 삭제  \* @pre 장르리스트가 초기화 되어있어야 됨  \* @post 장르리스트에 일치하는 Music이 있으면 삭제  \* @param inData 삭제하고자 하는 Music클래스  \* @return 삭제에 성공했으면 1, 실패했으면 0 리턴  \*/  void Delete(SimpleMusicType inData);  // 연산자 오버로딩  /\*\*  \* @brief 장르클래스 간에 == 연산자 오버로딩  \* @param obj 해당 클래스와 연산될 피연산자 클래스  \*/  bool operator== (GenreType &obj)  {  if (genre == obj.GetGenre())  return true;  else  return false;  }  /\*\*  \* @brief 장르클래스 간에 > 연산자 오버로딩  \* @param obj 해당 클래스와 연산될 피연산자 클래스  \*/  bool operator> (GenreType &obj)  {  if (genre > obj.GetGenre())  return true;  else  return false;  }  /\*\*  \* @brief 장르클래스 간에 < 연산자 오버로딩  \* @param obj 해당 클래스와 연산될 피연산자 클래스  \*/  bool operator< (GenreType &obj)  {  if (genre < obj.GetGenre())  return true;  else  return false;  }  /\*\*  \* @brief 장르클래스 간에 딥카피를 위한 = 연산자 오버로딩  \* @param obj 해당 클래스와 연산될 피연산자 클래스  \*/  GenreType& operator=(GenreType& obj)  {  genre = obj.GetGenre();  m\_List = obj.m\_List;  return \*this;  }  private:  string genre; ///< 장르명(primary key)  int m\_Length; ///< 속한 곡수  BinarySearchTree<SimpleMusicType> m\_List; ///< 속한 곡  };  MusicType.h  class MusicType  {  public:  /\*\*  \* 디폴트 생성자  \*/  MusicType()  {  song = "";  singer = "";  album = "";  genre = "";  }  /\*\*  \* 소멸자  \*/  ~MusicType() {}  /\*\*  \* @brief 곡명 가져오기  \* @pre 곡명이 설정되있어야 함  \* @post none.  \* @return 곡명  \*/  string GetSong()  {  return song;  }  /\*\*  \* @brief 가수명 가져오기  \* @pre 가수가 설정되있어야 함  \* @post none.  \* @return 가수  \*/  string GetSinger()  {  return singer;  }  /\*\*  \* @brief 작곡가 가져오기  \* @pre 작곡가가 설정되있어야 함  \* @post none.  \* @return 작곡가  \*/  string GetSongWriter()  {  return songwriter;  }  /\*\*  \* @brief 작사가 가져오기  \* @pre 작사가가 설정되있어야 함  \* @post none.  \* @return 작사가  \*/  string GetLyricWriter()  {  return lyricwriter;  }  /\*\*  \* @brief 앨범 가져오기  \* @pre 앨범이 설정되있어야 함  \* @post none.  \* @return 앨범  \*/  string GetAlbum()  {  return album;  }  /\*\*  \* @brief 장르 가져오기  \* @pre 장르가 설정되있어야 함  \* @post none.  \* @return 장르  \*/  string GetGenre()  {  return genre;  }  /\*\*  \* @brief 가사 가져오기  \* @pre 가사가 설정되있어야 함  \* @post none.  \* @return 가사  \*/  string GetLyrics()  {  return lyrics;  }  /\*\*  \* @brief 곡명 설정  \* @pre none.  \* @post 곡명 설정  \* @param inSong 설정할 곡명.  \*/  void SetSong(string inSong)  {  song = inSong;  }  /\*\*  \* @brief 가수명 설정  \* @pre none.  \* @post 가수명 설정  \* @param inSinger 가수명  \*/  void SetSinger(string inSinger)  {  singer = inSinger;  }  /\*\*  \* @brief 작곡가 설정  \* @pre none.  \* @post 작곡가 설정  \* @param inSongWriter 설정할 작곡가  \*/  void SetSongWriter(string inSongWriter)  {  songwriter = inSongWriter;  }  /\*\*  \* @brief 작사가 설정  \* @pre none.  \* @post 작사가 설정  \* @param inLyricWriter 설정할 작사가  \*/  void SetLyricWriter(string inLyricWriter)  {  lyricwriter = inLyricWriter;  }  /\*\*  \* @brief 앨범 설정  \* @pre none.  \* @post 앨범 설정  \* @param inAlbum 앨범  \*/  void SetAlbum(string inAlbum)  {  album = inAlbum;  }  /\*\*  \* @brief 장르 설정  \* @pre none.  \* @post 장르 설정  \* @param inGenre 장르  \*/  void SetGenre(string inGenre)  {  genre = inGenre;  }  /\*\*  \* @brief 가사 설정  \* @pre none.  \* @post 가사 설정  \* @param inLyrics 가사  \*/  void SetLyrics(string inLyrics)  {  lyrics = inLyrics;  }  /\*\*  \* @brief 노래 정보 설정  \* @pre none.  \* @post 노래 정보 설정  \* @param inSong 곡명  \* @param inSinger 가수명  \* @param inSongWriter 설정할 작곡가  \* @param inLyricWriter 설정할 작사가  \* @param inAlbum 앨범  \* @param inGenre 장르  \* @param inLyrics 가사  \*/  void SetRecord(string inSong, string inSinger, string inSongWriter, string inLyricWriter, string inAlbum, string inGenre, string inLyrics)  {  SetSong(inSong);  SetSinger(inSinger);  SetSongWriter(inSongWriter);  SetLyricWriter(inLyricWriter);  SetAlbum(inAlbum);  SetGenre(inGenre);  SetLyrics(inLyrics);  }  /\*\*  \* @brief 화면에 곡명 출력  \* @pre 곡명이 설정되있어야 함  \* @post none  \*/  void DisplaySongOnScreen()  {  cout << "\t곡명 : " << song << endl;  };  /\*\*  \* @brief 화면에 가수명 출력  \* @pre 가수명이 설정되있어야 함  \* @post none  \*/  void DisplaySingerOnScreen()  {  cout << "\t가수명 : " << singer << endl;  };  /\*\*  \* @brief 화면에 작곡가 출력  \* @pre 작곡가가 설정되있어야 함  \* @post none  \*/  void DisplaySongWriterOnScreen()  {  cout << "\t작곡가 : " << songwriter << endl;  };  /\*\*  \* @brief 화면에 작사가 출력  \* @pre 작사가가 설정되있어야 함  \* @post none  \*/  void DisplayLyricWriterOnScreen()  {  cout << "\t작사가 : " << lyricwriter << endl;  };  /\*\*  \* @brief 화면에 앨범 출력  \* @pre 앨범이 설정되있어야 함  \* @post none  \*/  void DisplayAlbumOnScreen()  {  cout << "\t앨범명 : " << album << endl;  };  /\*\*  \* @brief 화면에 장르 출력  \* @pre 장르가 설정되있어야 함  \* @post none  \*/  void DisplayGenreOnScreen()  {  cout << "\t장르 : " << genre << endl;  };  /\*\*  \* @brief 화면에 가사 출력  \* @pre 가사가 설정되있어야 함  \* @post none  \*/  void DisplayLyricsOnScreen()  {  cout << "\t가사 : " << lyrics << endl;  };  /\*\*  \* @brief 화면에 노래 정보 출력  \* @pre 노래 정보가 설정되있어야 함  \* @post none  \*/  void DisplayRecordOnScreen()  {  DisplaySongOnScreen();  DisplaySingerOnScreen();  DisplaySongWriterOnScreen();  DisplayLyricWriterOnScreen();  DisplayAlbumOnScreen();  DisplayGenreOnScreen();  DisplayLyricsOnScreen();  };  /\*\*  \* @brief 키보드로 곡명 설정  \* @pre none.  \* @post 곡명 설정  \*/  void SetSongFromKB();  /\*\*  \* @brief 키보드로 가수명 설정  \* @pre none.  \* @post 가수명 설정  \*/  void SetSingerFromKB();  /\*\*  \* @brief 키보드로 작곡가 설정  \* @pre none.  \* @post 작곡가 설정  \*/  void SetSongWriterFromKB();  /\*\*  \* @brief 키보드로 작사가 설정  \* @pre none.  \* @post 작사가 설정  \*/  void SetLyricWriterFromKB();  /\*\*  \* @brief 키보드로 앨범 설정  \* @pre none.  \* @post 앨범 설정  \*/  void SetAlbumFromKB();  /\*\*  \* @brief 키보드로 장르 설정  \* @pre none.  \* @post 장르 설정  \*/  void SetGenreFromKB();  /\*\*  \* @brief 키보드로 가사 설정  \* @pre none.  \* @post 가사 설정  \*/  void SetLyricsFromKB();  /\*\*  \* @brief 키보드로 노래 정보 설정  \* @pre none.  \* @post 노래 정보 설정  \*/  void SetRecordFromKB();  // 연산자 오버로딩  /\*\*  \* @brief 뮤직클래스 간에 == 연산자 오버로딩  \* @param obj 해당 클래스와 연산될 피연산자 클래스  \*/  bool operator== (MusicType &obj)  {  if (song == obj.GetSong())  return true;  else  return false;  }  /\*\*  \* @brief 뮤직클래스 간에 > 연산자 오버로딩  \* @param obj 해당 클래스와 연산될 피연산자 클래스  \*/  bool operator> (MusicType &obj)  {  if (song > obj.GetSong())  return true;  else  return false;  }  /\*\*  \* @brief 뮤직클래스 간에 < 연산자 오버로딩  \* @param obj 해당 클래스와 연산될 피연산자 클래스  \*/  bool operator< (MusicType &obj)  {  if (song < obj.GetSong())  return true;  else  return false;  }  /\*\*  \* @brief 뮤직클래스의 ostream시 << 연산자 오버로딩  \* @param out ostream시 할 out 종류  \* @param music out과 연산될 피연산자 뮤직클래스  \*/  friend void operator<<(ostream& out, MusicType& music);  /\*\*  \* @brief 뮤직클래스의 ofstream시 << 연산자 오버로딩  \* @param out ofstream시 할 out 종류  \* @param music out과 연산될 피연산자 뮤직클래스  \*/  friend void operator<<(ofstream& out, MusicType& music);  private:  string song; ///< 노래제목(primary key)  string singer; ///< 가수  string songwriter; ///< 작곡가  string lyricwriter; ///< 작사가  string album; ///< 앨범  string genre; ///< 장르  string lyrics; ///< 가사  };  SimpleMusicType.h  class SimpleMusicType  {  public:  SimpleMusicType();  ~SimpleMusicType();  /\*\*  \* @brief 재생횟수 증가  \* @post 재생횟수가 1 증가함  \*/  void AddNumberOfPlay(); // 재생횟수 추가  /\*\*  \* @brief 곡명 가져오기  \* @pre 곡명이 설정되있어야 함  \* @post none.  \* @return 곡명  \*/  string GetSong()  {  return song;  }  /\*\*  \* @brief 가수명 가져오기  \* @pre 가수명이 설정되있어야 함  \* @post none.  \* @return 가수명  \*/  string GetSinger()  {  return singer;  }  /\*\*  \* @brief 재생횟수 가져오기  \* @pre 재생횟수가 설정되있어야함  \* @return 재생횟수  \*/  int GetNumberOfPlay() // 재생횟수 가져오기  {  return numberofplay;  }  /\*\*  \* @brief 곡명 설정  \* @pre none.  \* @post 곡명 설정  \* @param inSong 설정할 곡명.  \*/  void SetSong(string inSong)  {  song = inSong;  }  /\*\*  \* @brief 가수명 설정  \* @pre none.  \* @post 가수명 설정  \* @param inSinger 설정할 가수명.  \*/  void SetSinger(string inSinger)  {  singer = inSinger;  }  /\*\*  \* @brief 노래 정보 설정  \* @pre none.  \* @post 노래 정보 설정  \* @param inSong 설정할 곡명  \* @param inSinger 설정할 가수명  \*/  void SetRecord(string inSong, string inSinger)  {  SetSong(inSong);  SetSinger(inSinger);  }  /\*\*  \* @brief 화면에 곡명 출력  \* @pre 곡명이 설정되있어야 함  \* @post none  \*/  void DisplaySongOnScreen()  {  cout << "\t곡명 : " << song << endl;  };  /\*\*  \* @brief 화면에 가수명 출력  \* @pre 가수명이 설정되있어야 함  \* @post none  \*/  void DisplaySingerOnScreen()  {  cout << "\t가수명 : " << singer << endl;  };  /\*\*  \* @brief 화면에 재생횟수 출력  \* @pre 재생횟수가 설정되있어야 함  \* @post none  \*/  void DisplayNumberOfPlayOnScreen()  {  cout << "\t재생횟수 : " << numberofplay << endl;  };  /\*\*  \* @brief 화면에 간단한 노래 정보 출력  \* @pre 노래 정보가 설정되있어야 함  \* @post none  \*/  void DisplayRecordOnScreen()  {  DisplaySongOnScreen();  DisplaySingerOnScreen();  };  // 연산자 오버로딩  /\*\*  \* @brief 뮤직클래스 간에 == 연산자 오버로딩  \* @param obj 해당 클래스와 연산될 피연산자 클래스  \*/  bool operator== (SimpleMusicType &obj)  {  if (song == obj.GetSong())  return true;  else  return false;  }  /\*\*  \* @brief 뮤직클래스 간에 > 연산자 오버로딩  \* @param obj 해당 클래스와 연산될 피연산자 클래스  \*/  bool operator> (SimpleMusicType &obj)  {  if (song > obj.GetSong())  return true;  else  return false;  }  /\*\*  \* @brief 뮤직클래스 간에 < 연산자 오버로딩  \* @param obj 해당 클래스와 연산될 피연산자 클래스  \*/  bool operator< (SimpleMusicType &obj)  {  if (song < obj.GetSong())  return true;  else  return false;  }  /\*\*  \* @brief 심플뮤직클래스의 ostream시 << 연산자 오버로딩  \* @param out ostream시 할 out 종류  \* @param simplemusic out과 연산될 피연산자 심플뮤직클래스  \*/  friend void operator<<(ostream& out, SimpleMusicType& simplemusic);  private:  string song; ///< 곡명(primary key)  string singer; ///< 가수  int numberofplay; ///< 재생횟수  }; |
| 1. 개발 환경 |

* Microsoft Visual Studio 2017 (C++)