2019년 1학기
자료구조 설계 중간보고서

Movie Database Application

제출일 : 2019년 5월 20일

담당교수	이영구 교수님
이름	우수진
Email	wsj1998@naver.com

1. 작품 요약

A. 작품의 필요성 (문제 정의)

시장조사전문 기업[1]에서 성인남녀 1,000명을 대상으로 '영화'에 관한 설문조사를 한 결과, 약 90%의 사람들이 영화 관람을 가장 쉽게 접근할 수 있는 문화생활이라고 답하였다. 이렇다 할 여가활동이 여전히 충분하지 못한 한국사회에서 영화 관람이 가장 대중적인 문화생활로 자리 잡고 있음을 확인시켜주는 결과이다.

이에 현대인들이 SNS, 사진, 블로그 등을 통해 자신의 일상을 기록하는 모습이 떠올랐다. 이제 영화도 우리 일상의 한 부분이 되어가고 있는 만큼, 이제는 이를 기록할 수 있는 서비스도 필요하다고 생각한다. 이에 'Movie Database Application'을 개발하게 되었다.

'Movie Database Application'은 영화이름, 배우, 개봉년도 등과 같은 주요 정보를 기입하여 사람들이 자신이 본 영화에 대해 기록을 남길 수 있도록 도와주는 프로그램이다. 원한다면 언제든지 세부사항들을 추가 및 삭제할 수 있다.

B. 작품의 요구 사항

프로젝트의 이름이 'Movie Database Application'인 만큼 이 프로그램을 통해서 사용자는 영화에 대한 정보를 관리할 수 있다. 우선 사용자는 원하는 영화를 추가할 수 있는데 이때 이름뿐만 아니라 영화에 출연한 배우의 이름, 등급 등과 같은 세부사항들도 함께 넣을 수 있다. 또한 사용자가 원한다면 언제든지 영화에 대한 모든 정보를 삭제하거나, 영화에서의 일부사항만 수정하는 것이 가능하다.

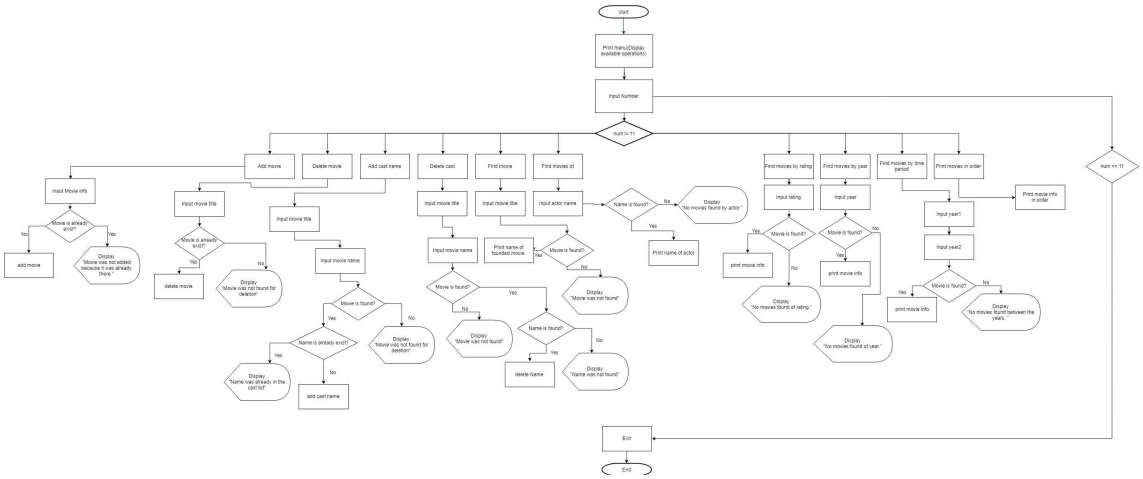
그리고 이렇게 추가된 영화는 배우이름, 개봉년도, 상영시간 등의 다양한 카테고리로 분류된다. 그리고 사용자는 영화 이름 말고도 원하는 카테고리에 따라 영화를 검색할 수 있다. 마지막으로는 그동안 사용자가 저장한 영화들을 한꺼번에 확인해 볼 수 있는 print기능도 제공한다.

---

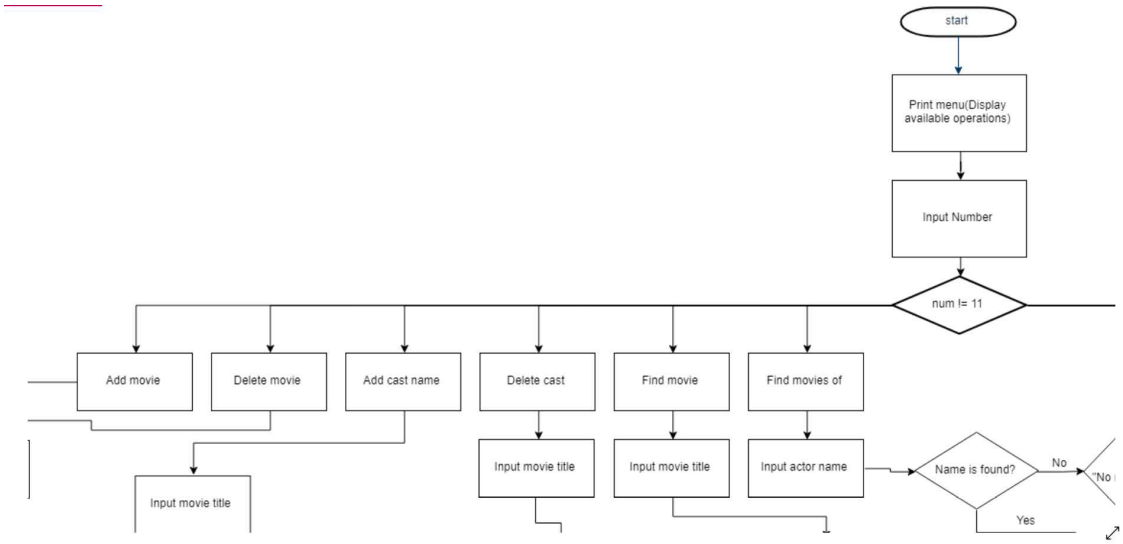
1) https://www.trendmonitor.co.kr/tmweb/trend/allTrend/detail.do?bIdx=1485&code=0303&trendType=CKOREA&prevMonth=&currentPage=2

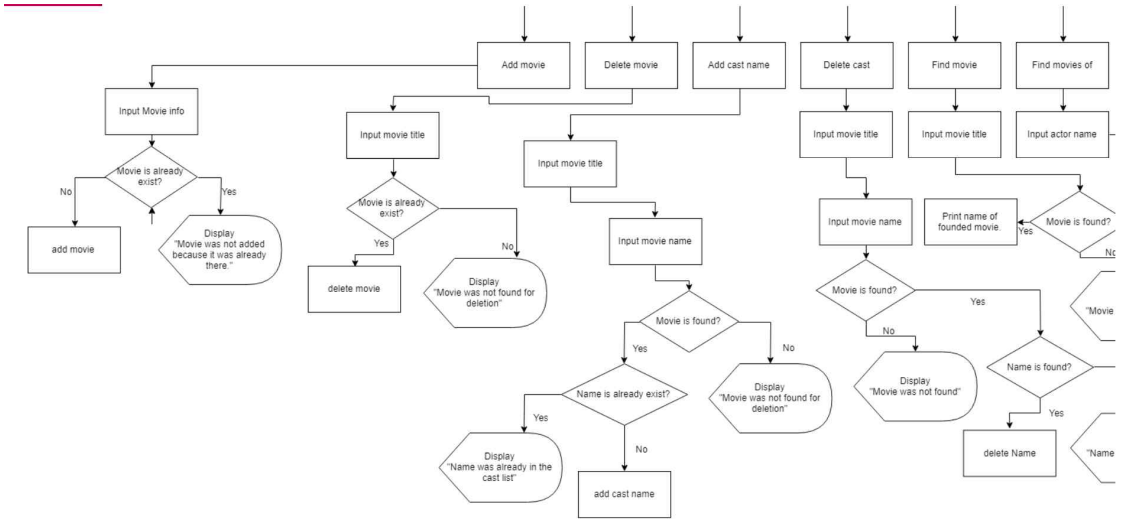
2. 작품 설계의 구체화

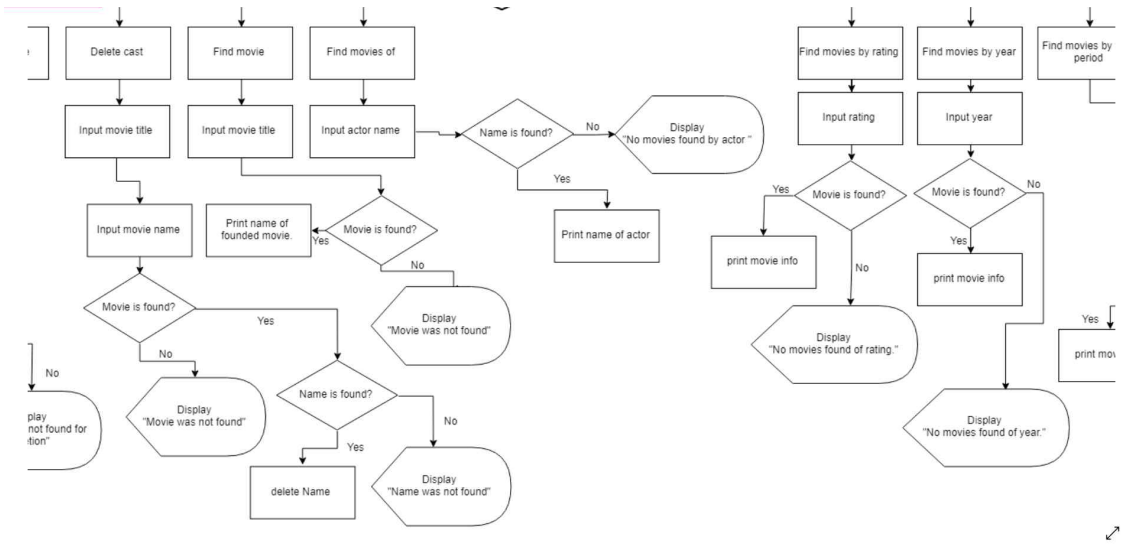
A. 작품의 기능 블록 다이어그램 (png파일을 첨부하였습니다.)

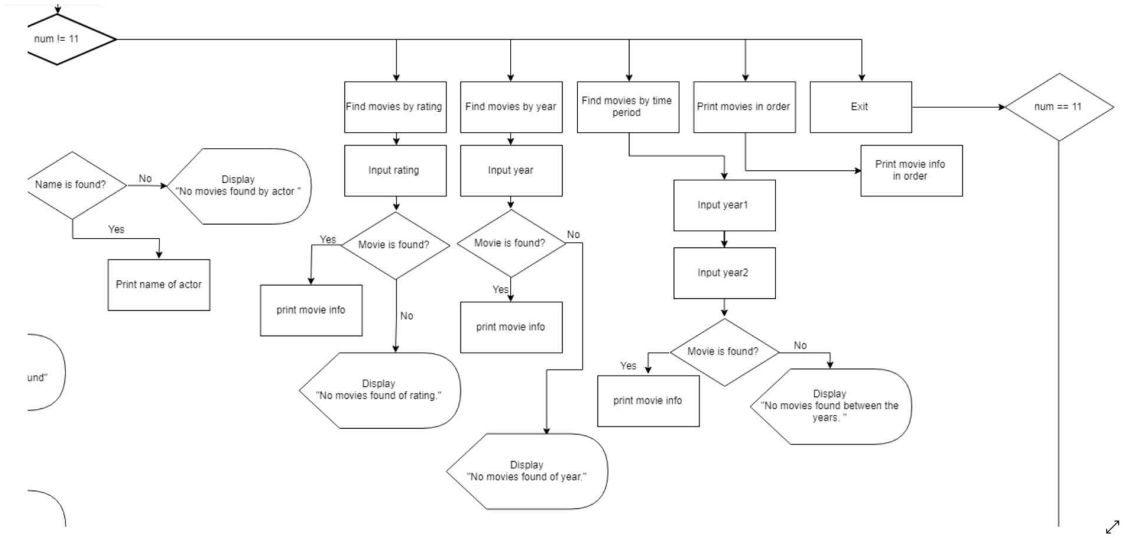


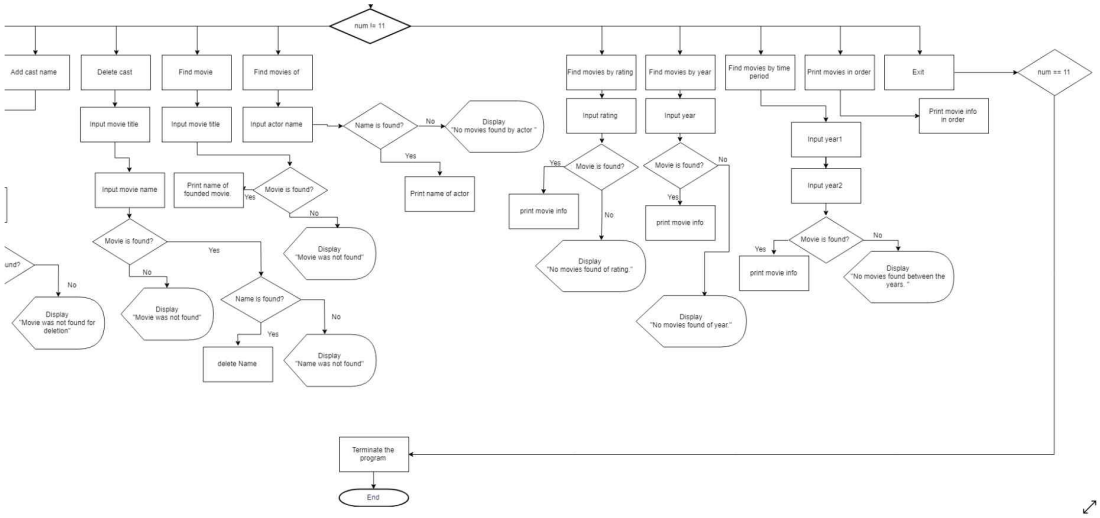
입출력과 각 기능을 Flow chart를 통해 나타내 보았다.




Add movie
Delete movie
Add cast name
Delete cast
Find movie
Find movies of
Input Movie info
Movie is already exist?
No
Yes
add movie
Display "Movie was not added because it was already there."
Input movie title
Movie is already exist?
Yes
No
delete movie
Display "Movie was not found for deletion"
Input movie title
Input movie name
Movie is found?
Yes
No
Name is already exist?
Yes
No
Display "Name was already in the cast list"
add cast name
Display "Movie was not found for deletion"
Input movie title
Input movie name
Movie is found?
No
Yes
Display "Movie was not found"
Name is found?
Yes
delete Name
Input movie title
Input actor name
Print name of founded movie.
Movie is found?
Yes
Movie is found?
Yes
No


Delete cast
Find movie
Find movies of
Input movie title
Input movie title
Input actor name
Name is found?
No
Display "No movies found by actor "
Yes
Print name of actor
Input movie name
Print name of founded movie.
Movie is found?
Yes
No
Movie is found?
Yes
No
Display "Movie was not found"
Display "Movie was not found"
Name is found?
Yes
No
delete Name
Display "Name was not found"
Find movies by rating
Find movies by year
Find movies by period
Input rating
Input year
Movie is found?
Yes
No
print movie info
Display "No movies found of rating."
Movie is found?
Yes
No
print movie info
Display "No movies found of year."


num != 11
Find movies by rating
Find movies by year
Find movies by time period
Print movies in order
Exit
num == 11
Name is found?
No
Display "No movies found by actor "
Yes
Print name of actor
Input rating
Input year
Print movie info in order
Input year1
Input year2
Movie is found?
Yes
No
print movie info
Display "No movies found of rating."
Movie is found?
Yes
No
print movie info
Movie is found?
Yes
No
print movie info
Display "No movies found between the years. "
Display "No movies found of year."



B. 설계 사양 목표

1) cdlist_node.h

(1) class cdlist_node

① Default constructor

cdlist_node(const Item& init_data = Item(), cdlist_node *init_next = NULL, cdlist_node *init_prev = NULL)	
기능:	class constructor
조건:	
결과:	parameter is initialized

② Accessor methods

Item & get_data();	
기능:	get the data item(non-const version)
조건:	
결과:	return data

const Item & get_data() const;	
기능:	get the data item(const version)
조건:	
결과:	return data

cdlist_node *get_next();	
기능:	get the next link (non-const version)
조건:	
결과:	return link_next

const cdlist_node* get_next() const;	
기능:	get the next link (const version)
조건:	
결과:	return link_next

cdlist_node* get_prev();	
기능:	Get the prev link (non-const version)
조건:	
결과:	return link_prev

const cdlist_node* get_prev() const;	
기능:	Get the prev link (const version)
조건:	
결과:	return link_prev

③ Mutator methods

void set_data(const Item& new_data);	
기능:	Set the data value
조건:	
결과:	data = new_data

void set_next(cdlist_node* new_link);	
기능:	Set the next link
조건:	
결과:	link_next = new_link

void set_prev(cdlist_node* new_link);	
기능:	Set the previous link
조건:	
결과:	link_prev = new_link

(2) Function to manipulate the circular doubly linked list

void list_clear(cdlist_node<Item> *head_ptr);	
기능:	Removes all elements from the list, leaving it with only the header node.
조건:	head_ptr is assumed to be non-NULL
결과:	Clear the list, leaving only the header node

void list_clear(cdlist_node<Item> *head_ptr, cdlist_node<Item> *tail_ptr);	
기능:	Removes all elements between two positions from the list.
조건:	The destination list is assumed to be empty. All pointers are assumed not NULL.
결과:	Clear all elements between two positions

void list_copy(cdlist_node<Item> *src_head_ptr, cdlist_node<Item> *dst_head_ptr);	
기능:	Copy all elements from linked list into another linked list. (Deep Copy of list structure only)
조건:	The destination list is assumed to be empty. Both pointers are assumed not NULL.
결과:	Copy the list (Deep Copy of list structure only)

void list_copy(cdlist_node<Item> *src_head_ptr, cdlist_node<Item> *src_tail_ptr, cdlist_node<Item> *dst_head_ptr);	
기능:	Copy a range of elements from one linked list into another linked list. (Deep Copy of list structure)
조건:	The destination list is assumed to be empty. All pointers are assumed not NULL.
결과:	Copy a group of contiguous elements (Deep Copy of list structure)

void list_insert(cdlist_node<Item> *target_ptr, const Item& obj);	
기능:	Inserts a new item into the linked list BEFORE the node pointed to by the target_ptr parameter.
조건:	target_ptr is assumed not NULL.
결과:	Insert a new element into the linked list.

void list_insert_after(cdlist_node<Item> *target_ptr, const Item& obj);	
기능:	This method inserts a new element after the node pointed to by argument target pointer.
조건:	target_ptr is assumed not NULL.
결과:	Inserts a new element into the list immediately after a given target pointer.

void list_remove(cdlist_node<Item>* target_ptr);	
기능:	Removes a given element from the list.
조건:	target_ptr is assumed not NULL.
결과:	Removes an element from the list

std::size_t list_length(cdlist_node<Item> *head_ptr);	
기능:	Returns the length of this list.
조건:	head_ptr is assumed to be non-NULL.
결과:	Get the length of the list

std::size_t list_length(cdlist_node<Item> *head_ptr, cdlist_node<Item> *tail_ptr);	
기능:	Counts all elements between two positions from the list
조건:	
결과:	Get the number of elements between two positions.

cdlist_node<Item>* list_search(cdlist_node<Item> *head_ptr, const Item& obj);	
기능:	Finds and returns the pointer to the node containing a given object.
조건:	
결과:	Find an element in the list

const cdlist_node<Item>* list_search(const cdlist_node<Item> *head_ptr, const Item& obj);	
기능:	Finds and returns constant pointer to the node containing a given object.
조건:	
결과:	Find an element in the list (const version)

cdlist_node<Item>* list_locate(cdlist_node<Item>* head_ptr, const Item& obj);	
기능:	Finds the location where a new element should go in order to keep an increasing sorted order. Returns the pointer to this location.
조건:	
결과:	Get the position where an element should go in the order

2) Name.h

(1) class Name

Name(string firstname_in = "NONE", string lastname_in = "NONE");	
기능:	Constructor based on the a first name and a last name.
조건:	
결과:	parameter is initialized

bool operator==(const Name& src) const;	
기능:	Determines if the argument is not lexicographically equal to this name.
조건:	
결과:	Return true if the source name is equal to this name, or false otherwise

bool operator!=(const Name& src) const	
기능:	Compare names for non-equality
조건:	
결과:	Return true if the source name is not equal to this name, or false otherwise.

bool operator<(const Name& src) const;	
기능:	Determines if the argument name is lexicographically less than this name.
조건:	
결과:	Return true if the source name is less than this name, or false otherwise.

bool operator>(const Name& src) const;	
기능:	Determine if the argument is lexicographically greater than this name.
조건:	
결과:	Return true if the source name is less than this name, or false otherwise.

bool operator<=(const Name& src) const;	
기능:	Determines if the argument is lexicographically less or equal to this name.
조건:	
결과:	Return true if the source name is less or equal to this name, or false otherwise.

bool operator>=(const Name& src) const;	
기능:	Determines if the argument is lexicographically greater or equal to this name.
조건:	
결과:	Return: true if the source name is greater or equal to this name, or false otherwise.

string get_key() const;	
기능:	Returns a string representing the key in this name object as the combination of lastname followed by first name.
조건:	
결과:	Return key for this name

string get_fullname() const;	
기능:	Returns a string representing the full name in this name.
조건:	
결과:	Returns the full name as string

const string& get_lastname() const;	
기능:	Returns the last name part of this name.
조건:	
결과:	Returns the last name field

const string& get_firstname() const;	
기능:	Returns the first name field
조건:	
결과:	Returns the first name part of this name.

friend istream& operator>>(istream& input, Name &target);	
기능:	Read the value for this name.
조건:	
결과:	Return reference to the argument input stream.

friend ostream& operator<<(ostream& output, const Name &src);	
기능:	Write a name to an output stream.
조건:	
결과:	Return reference to the argument output stream.

3) List.h - #include cdlist_node.h"

(1) class List_Iterator

bool has_data() const;	
기능:	Determines if the iterator has any data left.
조건:	
결과:	Determines if the iterator has data to return from its current position

Item& get_data();	
기능:	Returns a reference to the object stored at the node currently being pointed to by the iterator (current iterator position).
조건:	
결과:	Returns the data stored as the current position for this iterator.

void next();	
기능:	Moves the iterator from its current position to the next one.
조건:	the iterator is declared as a forward iterator.
결과:	Advances the iterator to the next valid position on the list.

void prev();	
기능:	Moves the iteration from its current position to the previous one.
조건:	the iterator is declared as a backward iterator.
결과:	Advances the iterator to the previous position with respect to the current position of the iterator on the list.

List_Iterator()	
기능:	Private default constructor, added mainly for convenience so we can easily declare iterator variables
조건:	
결과:	This constructor does not produce valid iterators.

List_Iterator(cdlist_node<Item> *init_head, cdlist_node<Item> *init_tail, bool go_forward = false);	
기능:	Creates a new iterator based on the first and last element over which the iterator shall occurs.
조건:	Both argument are assumed to be non-NULL.
결과:	Constructor based on the first and last element over which the iterator shall occurs. Both argument are assumed to be non-NULL.

(2) class List

List();	
기능:	Default constructor
조건:	
결과:	create an empty list

List(const List& L);	
기능:	Copy constructor
조건:	
결과:	create a new list by making a deep copy of another list L.

~List();	
기능:	Destructor
조건:	
결과:	clear all memory allocated to the list.

const List& operator=(const List& L);	
기능:	Copy assignment operator
조건:	
결과:	re-assign the value of the list by making a deep copy of another list L.

bool is_empty() const	
기능:	Determines if the list is an empty list
조건:	An empty list has only one with its prev and next links pointing to itself.
결과:	Returns true if the list is empty, or false otherwise.

void make_empty();	
기능:	Makes the list an empty list
조건:	
결과:	Removes all elements from the list, making it an empty list.

List_Iterator<Item> first() const	
기능:	Returns an iterator to the first element of the list
조건:	
결과:	Returns iterator object with current position at first element of list

List_Iterator<Item> last() const	
기능:	Returns an iterator to the last element in the list
조건:	
결과:	Returns iterator object with current position at last element of list

List_Iterator<Item> find(const Item& obj) const	
기능:	Finds an element in the list, and returns an iterator to it.
조건:	If the element is found, the iterator will have its current position set to the node where the element resides.
결과:	Finds an element in the list and returns an iterator to it.

void insert(const Item& obj);	
기능:	Inserts a new element into the list at its proper position based on the sorted order.
조건:	
결과:	Insert a new element at the proper location in the list so that the increasing order in the list is maintained.

bool erase(const Item& obj);	
기능:	Removes an element from the list, and keep sorted order.
조건:	
결과:	If the element is found and deleted returns true, or returns false.

size_type size() const	
기능:	Returns the size of the list
조건:	
결과:	Returns the number of elements in the list.

void print_forward(ostream& out) const	
기능:	Prints the contents of the list in the forward direction.
조건:	This method is mainly for debugging purposes.
결과:	Prints the contents of the list in the forward direction.

void print_backward(ostream& out) const	
기능:	Prints the contents of the list in the backward direction.
조건:	This method is mainly for debugging purposes
결과:	Prints the contents of the list in the backward direction.

4) NameList.h - #include "Name.h", #include "List.h"

(1) class NameList

NameList();	
기능:	Default constructor
조건:	
결과:	create a new name list with no names in it.

NameList(const NameList& L);	
기능:	Copy constructor
조건:	
결과:	create a deep copy of a given source name list.

const NameList& operator=(const NameList& L);	
기능:	Copy assignment operator
조건:	
결과:	create a deep copy of a given source name list.

void add(const Name& name);	
기능:	Adds new name into the list of names at its proper location based on increasing lexicographical order.
조건:	
결과:	Adds a new name to the list

bool find(const Name& name) const;	
기능:	Determines if a name is currently being stored in the name list.
조건:	
결과:	Return true if the name is in the list, or false otherwise.

bool erase(const Name& name);	
기능:	Removes a name from the list of names, if the name is present.
조건:	
결과:	Return true if the element was found and deleted, or false otherwise

int size() const;	
기능:	Returns the size of the name list
조건:	
결과:	name list의 size를 반환한다.

bool is_empty() const;	
기능:	Determines if the list is empty
조건:	
결과:	Returns the number of elements in the name list.

friend ostream& operator<<(ostream& out, const NameList& L);	
기능:	Prints all the names to an output stream. Each name is printed in a new line.
조건:	
결과:	Return reference to the output stream.

friend istream& operator>>(istream& in, NameList& L);	
기능:	Reads a list of names from an input stream.
조건:	
결과:	Reads a group of names from the input stream

5) Movie.h - #include "NameList.h"

(1) class Movie

Movie(const string& title_in = "NONE", int year_in = 0, int duration_in = 0, const string& rating_in = "NR", const NameList& castlist_in = NameList());	
기능:	Default constructor
조건:	
결과:	creates a new movie from a title, release year, duration, rating and cast name list.

string get_key() const	
기능:	Returns the key for this movie
조건:	
결과:	Returns the key for this movie, which is the movie title.

string get_title() const	
기능:	Returns the title for this movie.
조건:	
결과:	Returns the string representing title for this movie.

int get_year() const	
기능:	Returns the year for this movie
조건:	
결과:	Returns the year of release for this movie.

int get_duration() const	
기능:	Returns the duration for this movie.
조건:	
결과:	Returns the duration (in minutes) for this movie

string get_rating() const	
기능:	Returns the string representing the rating for this movie.
조건:	
결과:	Returns the rating for this movie

const NameList& get_cast_list() const	
기능:	Returns the Namelist with the cast for this movie
조건:	
결과:	Returns the name list for the cast of this movie.

bool is_cast_member(const Name& name) const	
기능:	Determines if a name is in the cast member list
조건:	
결과:	Return true if the name is in the cast list, or false otherwise.

void set_title(const string& new_title);	
기능:	Sets the title for this movie
조건:	
결과:	Sets a new title for this movie.

void set_year(int new_year);	
기능:	Sets the year for this movie
조건:	
결과:	Sets a new year of release for this movie.

void set_duration(int new_duration);	
기능:	Sets the duration for this movie
조건:	the duration is at least 1 minute.
결과:	Sets a new duration for this movie. The function

void set_rating(const string& new_rating);	
기능:	Sets the rating for this movie
조건:	
결과:	Sets a new rating for this movie. The function asserts that the rating is valid.

void set_cast_list(const NameList& L);	
기능:	Sets the cast name list for this movie
조건:	
결과:	Sets the cast name list for this movie.

bool add_cast_member(const Name& name);	
기능:	Adds a new name to the cast list, if not already there
조건:	
결과:	Return true if the name is added, or false if the name is already there.

bool erase_cast_member(const Name& name);	
기능:	Removes a cast name from the cast list, if the name is there.
조건:	
결과:	Return true if the name is found and deleted, or false if the name is not found.

bool operator==(const Movie& src) const	
기능:	Determines if another movie has a key equal to this movie's key
조건:	
결과:	Return true if the key of the source movie is equal to this movie's key, or false otherwise.

bool operator!=(const Movie& src) const	
기능:	Determines if another movie has a key not equal to this movie's key
조건:	
결과:	Return true if the key of the source movie is not equal to this movie's key, or false otherwise.

bool operator<(const Movie& src) const	
기능:	Determines if another movie has a key less than this movie's key
조건:	
결과:	Return true if the key of the source movie is less than this movie's key, or false otherwise.

bool operator>(const Movie& src) const	
기능:	Determines if another movie has a key greater than this movie's key
조건:	
결과:	Return true if the key of the source movie is greater than this movie's key, or false otherwise.

bool operator<=(const Movie& src) const	
기능:	Determines if another movie has a key less than or equal to this movie's key
조건:	
결과:	Return true if the key of the source movie is less than or equal to this movie's key, or false otherwise.

bool operator>=(const Movie& src) const	
기능:	Determines if another movie has a key greater than or equal to this movie's key
조건:	
결과:	Return true if the key of the source movie is greater than or equal to this movie's key, or false otherwise.

friend istream& operator>>(istream& in, Movie& obj);	
기능:	Reads this movie info from an input stream
조건:	
결과:	Return the reference to the input stream.

friend ostream& operator<<(ostream& out, const Movie& obj);	
기능:	Writes this movie info to an output stream
조건:	
결과:	Return the reference to the output stream.

6) queue.h - #include "cdlist_node.h"

(1) class queue

queue();	
기능:	Default constructor
조건:	
결과:	Creates a new empty queue by making the front and read be two dummy headers.

queue(const queue& Q);	
기능:	Creates a new deep copy of the queue passed as parameter.
조건:	
결과:	copies all the elements between the front and the rear of the source queue.

~queue();	
기능:	Destructor
조건:	
결과:	erases all the nodes between the front and the rear of the queue, and then deletes the front and rear pointer.

const queue& operator=(const queue& Q);	
기능:	Makes a deep copy of the queue passed as parameter.
조건:	
결과:	copies all the elements between the front and the rear of the source queue.

Item& front() const	
기능:	Returns the element at the head of the queue
조건:	
결과:	Returns a reference to the element at the front of the queue.

void enqueue(const Item& obj);	
기능:	Insert a new element at the tail of the queue
조건:	
결과:	Inserts a new element into the queue by adding the new element in before the queue rear pointer, and increase the queue size by one.

void dequeue();	
기능:	Removes the element at the head of the queue
조건:	
결과:	Removes the element at the front of the queue, and decrease the queue size by one.

bool is_empty() const	
기능:	Determines if the queue is empty
조건:	An empty queue has size zero and the next pointer of the q_front is equal to q_rear.
결과:	Determines if the queue is empty.

void make_empty();	
기능:	Makes the queue an empty queue by remove all elements.
조건:	
결과:	Make the queue empty.

size_type size() const	
기능:	Returns the size of the queue
조건:	
결과:	Returns the size of the queue.

void print_queue(ostream& out);	
기능:	Prints the contents of this queue.
조건:	The Item data type has overloaded the operator <<
결과:	Prints the contents of this queue.

7) BinarySearchTree.h - #include "queue."

(1) class BST_inorder_iterator

bool has_data() const;	
기능:	Determines if the iterator has data to be returned from the current position in the tree.
조건:	
결과:	Output is true if the iterator has data to be returned or false otherwise.

BSTData& get_data() const	
기능:	Returns the data stored at the current position. This method throws an assertion if there is no data to be returned from the current node
조건:	
결과:	Output is data at the current node in the tree being traversed.

void next();	
기능:	Advances the iterator to the next node in the tree with respect to the in-order traversal.
조건:	there is a next node.
결과:	Advances the iterator to the next node in the tree with respect to the in-order traversal.

void reset();	
기능:	Resets the iterator to its original state.
조건:	
결과:	Resets the iterator to its original initial position.

BST_inorder_iterator(BSTNodePtr the_root = NULL);	
기능:	constructor
조건:	argument is not NULL
결과:	creates a new BST_inorder_iterator class, that can be used to perform an in-order tree traversal.

void queue_filler(BSTNodePtr node, queue<BSTNodePtr>& Q);	
기능:	queue filler
조건:	
결과:	add tree node ptr in a queue following an in-order tree traversal.

(2) class BST_preorder_iterator

bool has_data() const	
기능:	Determines if the iterator has data to be returned from the current position in the tree.
조건:	
결과:	Outputis true if the iterator has data to be returned or false otherwise.

BSTData& get_data() const	
기능:	Returns the data stored at the current position. This method throws an assertion if there is no data to be returned from the current node
조건:	
결과:	Output is data at the current node in the tree being traversed.

void next();	
기능:	Advances the iterator to the next node in the tree with respect to the in-order traversal. No action is taken if there is not a next node.
조건:	there is a next node.
결과:	Advances the iterator to the next node in the tree with respect to the pre-order traversal.

void reset();	
기능:	Resets the iterator to its original state.
조건:	
결과:	Resets the iterator to its original initial position.

BST_preorder_iterator(BSTNodePtr the_root = NULL);	
기능:	constructor
조건:	the argument is not NULL.
결과:	creates a new BST_preorder_iterator that can be used to perform a pre-order tree traversal.

void queue_filler(BSTNodePtr node, queue<BSTNodePtr>& Q);	
기능:	queue filler
조건:	
결과:	add tree node in a queue following an pre-order tree traversal.

(3) class BinarySearchTree

BinarySearchTree();	
기능:	Default constructor
조건:	
결과:	Creates an empty binary search tree, with a NULL root node and zero elements.

BinarySearchTree(const BinarySearchTree& T);	
기능:	Copy Constructor
조건:	
결과:	Creates a new deep copy of the binary search tree passed as argument.

~BinarySearchTree();	
기능:	Destructor
조건:	
결과:	Makes the BST an empty tree by removing all the nodes, including the tree.

const BinarySearchTree& operator=(const BinarySearchTree& T);	
기능:	Copy assignment operator
조건:	
결과:	Makes a deep copy of the source BST.

bool is_empty() const	
기능:	Determines if the tree is empty
조건:	
결과:	Returns true if the tree is empty, or false otherwise. An empty tree has a NULL root.

void make_empty();	
기능:	Makes the tree an empty tree
조건:	An empty tree has a NULL root.
결과:	Makes the BST an empty BST by erasing all the nodes, including the root.

void insert(const BSTData& obj);	
기능:	Inserts a new node in the binary search tree
조건:	This function must rely on the insert_aux function to use recursion to insert the new element at the proper subtree.
결과:	insert the new element at the proper subtree.

bool erase(const DataKey& key);	
기능:	If multiple copies of the same key exist, then the first one that is found is deleted.
조건:	If multiple copies of the same key exist, then the first one that is found is deleted.
결과:	Removes an element from the binary search tree based on its key

BST_preorder_iterator<BSTData, DataKey> find(const DataKey& key) const	
기능:	Finds an element in the BST based on its key, and returns an in-order iterator in which the first element is the element being searched
조건:	This function MUST rely on function find_aux to recursively search the element with the target key.
결과:	Output is iterator to the node where the element with the argument key is found, or an empty iterator.

int size() const	
기능:	Returns the number of items in the tree
조건:	
결과:	Returns the size of the tree as the number of elements it currently holds.

int height() const	
기능:	Returns the height of the tree
조건:	rely on function height_aux() to recursively compute the height of the tree.
결과:	Output is the height of the tree.

BST_inorder_iterator<BSTData, DataKey> get_in_order() const	
기능:	Returns an in-order tree iterator from the root
조건:	
결과:	Creates a new in-order iterator with all the nodes in the tree.

BST_inorder_iterator<BSTData, DataKey> get_in_order(const DataKey& key) const	
기능:	Returns an in-order tree iterator from a given node
조건:	If more that one exists, then the one nearest to the root is found.
결과:	Create a new in-order iterator with all the nodes in the subtree rooted at the node with holding a key value equal to the paramter.

void print_tree(ostream& out);	
기능:	Utility function used to print the nodes in the tree. This routing is can be used for debugging purposes.
조건:	relies of the auxiliary function print_tree_aux().
결과:	print the elements in the tree.

void insert_aux(const BSTData& obj, BSTNode<BSTData> * & node);	
기능:	insert a new elements on a BST tree. After the insert is completed, the size of the BST must be incremented by one.
조건:	The function MUST use recursion to find the right place to insert the node, according to the insertion algorithm for Binary Search Trees.
결과:	auxiliary insert routine

BSTNode<BSTData> *find_aux(const DataKey& key, BSTNode<BSTData> *node) const	
기능:	find the pointer to the BSTNode that contains an element with a key equal to the argument key.
조건:	This function MUST use recursion to find the node with the argument key.
결과:	Output is pointer to the first occurrence of a node containing a data element with a key equal to the argument key.

BSTNode<BSTData> *find_min_aux(BSTNode<BSTData> *node) const	
기능:	find a pointer to the node with the minimum key value on a tree. The functions receives as parameter a pointer to the root of the tree.
조건:	
결과:	auxiliary find maximum routine

bool erase_aux(const DataKey& key, BSTNode<BSTData> * & node);	
기능:	remove an node from the tree.
조건:	If more than one node exist with the given key, then the first one found is the one deleted.
결과:	auxiliary erase routine

int height_aux(BSTNode<BSTData> *node) const	
기능:	compute the height of a tree.
조건:	The function MUST use recursion to traverse the tree, and compute the height.
결과:	Output is height of the sub-tree.

void print_tree_aux(ostream& out, BSTNode<BSTData> *node, int n_space);	
기능:	print the elements in the tree
조건:	The parameter n_space is used to print blank spaces before printing the data contained on the root node.
결과:	auxiliary print tree routine

8) DataManager.h - #include "List.h", #include "BinarySearchTree.h", #include "Name.h", #include "NameList.h", #include "Movie.h"

(1) class DataManager

DataManager();	
기능:	Default Constructor
조건:	
결과:	creates an empty binary search tree (BST) to hold up the information about the movies.

int add_movie(const Movie& movie);	
기능:	Adds a new movie to the movie database, but only if the movie is not already there.
조건:	the movie is not already there.
결과:	Return 'OK', if the movie was added, or return 'MOVIE_WAS_THERE', if the movie was already in the movie list

int add_new_cast(const string& title, const Name& name);	
기능:	Adds a new member to the cast list of a given movie
조건:	
결과:	Return 'OK', if the new cast name is added. If no movie with the argument title is found, return 'MOVIE_NOT_FOUND'. if cast name was already in the cast list for the movie return 'NAME_WAS_THERE'

int delete_movie(const string& title);	
기능:	searches for a movie with a given title, and if found, deletes that movie.
조건:	
결과:	If the movie is deleted, return 'OK'. If the movie was not found in the list, return 'MOVIE_NOT_FOUND'.

int delete_cast(const string& title, const Name& name);	
기능:	Deletes a cast name from a movie
조건:	The function must first find the movie, and then delete the name from the cast list.
결과:	If the cast name is deleted, return 'MOVIE_NOT_FOUND'. if no movie with the argument title was found, return 'OK'. if the cast name was not found in the list, return 'NAME_NOT_FOUND'.

int find_movie(const string& title, Movie& movie) const	
기능:	Finds movie information about a given movie
조건:	
결과:	The function must return the code MOVIE_NOT_FOUND if no movie with the given title is found.

List<Movie> find_movies_of(const Name& name) const	
기능:	Finds movie information about movies made by a particular actor.
조건:	
결과:	Returns a list with all the movies found, or an empty list if the actor does not appears in the cast for any movie.

List<Movie> find_movies_rated(const string& rating) const	
기능:	Finds movie information about movies with a given rating
조건:	
결과:	It returns a list with all the movies found, or an empty list if no movie has the given rating.

List<Movie> find_movies_released(int year) const	
기능:	Finds movie information about movies made a given year
조건:	
결과:	It returns a list with all the movies found, or an empty list if no movie was released on the given year.

List<Movie> find_movies_between(int year1, int year2) const	
기능:	This function is used to find all the movies released between two years: year1 and year2.
조건:	All movies for which its release year Y, satisfies: year1 <= Y <= year2, will go in the result.
결과:	Returns a list with all the movies found, or an empty list if no movie was released during the given time period.

void print_movies_forward(ostream& out) const	
기능:	Prints all movies based on alphabetical order of the movie title.
조건:	
결과:	Print list of movies in order

void print_tree_height() const	
기능:	prints the height of the tree.
조건:	
결과:	the height of the tree is printed.

C. 상세 설계 후보안

1) "cdlist_node.h"

: 이 파일은 class cdlist_node의 선언을 포함한다. 그리고 이 파일에 활용된 자료구조와 함수들은 class List_Iterator, class List, class queue에서 활용된다.

(1) class cdlist_node

이 class는 circular-doubly linked list의 node를 나타낸다. 또한 Sorted circular doubly linked list를 구현한 함수도 포함하고 있다. 이때 node는 제약조건 아래에서 삽입되거나 삭제되며 이로써 특정 순서가 유지된다.

template parameter Item은 node에 저장될 object의 type이다. 각 Item의 상대적

인 순서를 결정하기 위해 node에 저장될 object의 type은 relational comparison operator를 오버로드한 것으로 가정한다.

2) "Name.h"
: 이 파일은 name data type의 선언을 포함한다.
이 파일의 내용은 class NameList, class DataManager에서 활용된다.

3) "List.h" - #inlcude "cdlist_node.h"
: 이 파일은 List container class에 대한 interface 선언을 포함하고 있다. 이 List는 Circular Doubly Linked List toolkit과 관련된 코드를 사용하여 Sorted Circular Doubly Linked List로 구현된다.
이 파일에 활용된 자료구조와 함수들은 class Name, class NameList에서 활용된다.

(1) class List_Iterator
class List_Iterator는 List container class에 대한 external iterator를 구현한다. 이 iterator는 리스트의 다른 노드들에 대해 access, traverse할 때 사용된다.
Item type 변수는 리스트에 저장될 element들의 type을 나타낸다.

(2) class List
이는 Sorted Circular Doubly Linked List로 구현된 List container class이다. Item type parameter는 리스트에 저장될 element들의 type을 나타낸다.

4) "NameList.h" - #inlcude "Name.h", #inlcude "List.h"
: 이 파일에는 list of names에 관한 interface 선언이 포함되어있으며, name을 저장하기 위해 Sorted Doubly Linked List class container가 사용된다.
이 파일에 활용된 자료구조와 함수들은 class Movie, class DataManager에서 활용된다.

(1) class NameList
이 class는 name의 Sorted List를 구현하기위해 사용된다. name의 저장을 위해서는 Sorted Doubly Linked List class container가 사용된다.
이 class는 default constructor가 Linked list에 대해 destructor를 올바르게 호출하고, 이 경우 List내의 element를 삭제해야 하므로 destructor가 구현되지 않는다.

5) "Movie.h" - #include "NmaeList.h"
이 파일은 class Movie에서 구현되는 movie data type의 interface선언이 포함되어 있다. 이 파일에 활용된 자료구조와 함수들은 class DataManager에서 활용된다.

(1) class Movie

이 class는 movie data type을 구현하는데 사용되는 class이다.

default destructor가 각 data member의 destructor를 올바르게 호출하고, NameList field에서 element를 삭제하기 때문에 이 class는 destructor를 구현하지 않는다.

6) "queue.h" - #inlcude "cdlist_node.h"

이 파일은 queue container class를 위한 interface의 선언을 포함한다. 이 class는 circular linked list toolkit으로 구현된다. 그러나 큐 자체는 circular도, sorted 형태도 아니다. 이 파일에 활용된 자료구조와 함수들은 class BST_Inorder_Iterator, class DataManager에서 활용된다.

(1) class queue

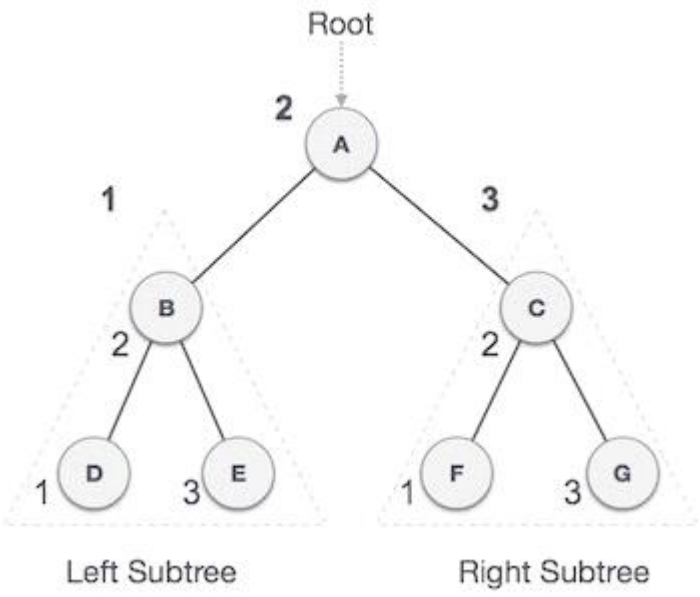
queue class container이다. 이 class는 queue data structure를 구현한다. Item Type parameter는 queue에 저장될 데이터 타입을 나타낸다.

7) "BinarySearchTree.h" - #inlcude "queue.h"

이 파일은 Binary Search Tree(BST)를 위한 interface 선언을 포함한다.

(1) class BST_inorder_iterator

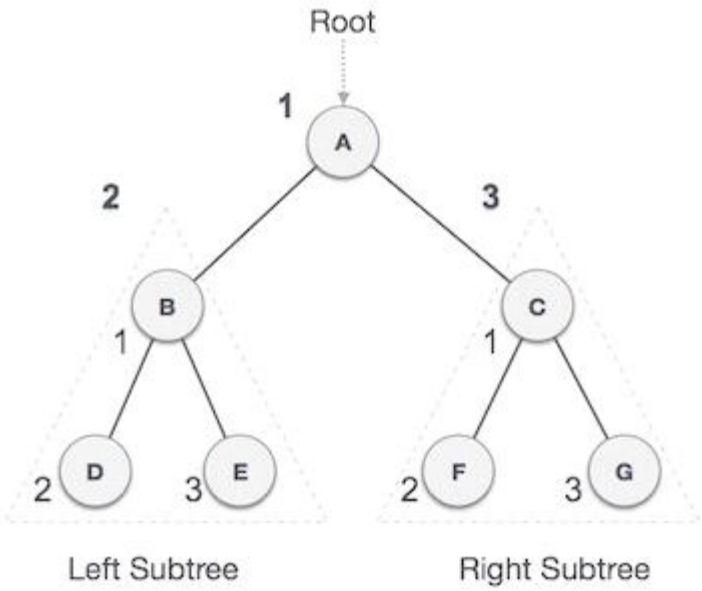
Binary Search Tree In-order Iterator class는 binary search tree의 inorder tree traversal을 수행하기위해 사용된다. BSTData는 search tree에 저장되는 data를 나타낸다. 그리고 DataKey type parameter는 tree에 저장될 object의 key를 나타낸다.



<Inorder Traversal>

(2) class BST_preorder_iterator

Binary Search Tree Pre-order Iterator class는 binary search tree의 preorder tree traversal을 수행하는데 사용된다. BSTData는 search tree에 저장될 data를 나타낸다. DataKey type parameter는 tree에 저장될 object의 key를 나타낸다.



<preorder traversal>

(3) class BinarySearchTree

Binary Search Tree class이다. 여기서 BSTData는 search tree에 저장될 data를 나타낸다. 그리고 DataKey type parameter는 tree에 저장될 object의 key를 나타낸다.

- Member parameter info

Movie
- title: string
- year: int
- duration: int
- rating: string
- castlist: NameList

Name
- lastname: string
- firstname: string

(1) Add movie

우선 BST에 사용자가 저장하려는 영화가 이미 존재하는지 확인한다. 여기서는 movie title이라는 key를 이용하여 이에 부합하는 값을 찾는다. 이때 recursion 방식을 이용한다. tree에서 주어진 object를 찾으면 pre-order tree iterator가 반환된다. 반면에 찾지 못하면 empty iterator가 반환된다. 그리고 이 정보를 통해 iterator가

current position에서 반환할 data를 가지고 있는지를 확인함으로써 영화의 존재 여부를 확인해준다.(이때 Queue 자료구조가 사용된다.) 그리고 사용자가 입력하려는 영화가 존재하지 않는다면 Binary Search Tree에 recursion을 사용해서 Movie정보를 저장해준다.

* 여기서 사용되는 Queue는 node들을 순서대로 가지고 있으며, iterator에 대한 current node는 queue의 front에 위치한다. Queue가 empty상태라면 이는 iterator에 data가 없음을 의미한다.
⇨ 사용된 자료구조: Binary Search Tree, List(배우 이름 저장), Queue

(2) Delete movie

우선 BST에 사용자가 저장하려는 영화가 이미 존재하는지 확인한다. 여기서는 movie title이라는 key를 이용하여 이에 부합하는 값을 찾는다. 이때도 recursion방식이 이용된다. tree에서 주어진 object를 찾으면 pre-order tree iterator가 반환되고, 찾지 못하면 empty iterator가 반환된다. 그리고 이 정보를 가지고 iterator가 current position에서 반환할 data를 가지고 있는지 확인함으로써 영화의 존재 여부를 확인해준다. (이때 Queue 자료구조가 사용된다.) 그리고 영화가 존재한다면 BST에서 element를 recursive하게 제거해준다.
⇨ 사용된 자료구조: Binary Search Tree, List(배우 이름 저장), Queue

* 위에서 이미 언급하여 반복되는 내용은 일정부분 생략하여 아래 내용을 작성 하였습니다.

(3) Add cast

이번에는 movie의 title key를 이용하여 BST에서 검색을 진행해준다. 마찬가지로 검색 결과 pre-order tree iterator가 반환된다. 그리고 iterator에 data가 있다면 castlist에 사용자가 입력하고자 하는 배우의 name이 circular doublye linked list에 있는지 검색한다. 없다면 name을 삽입하되, 삽입 후에도 list는 sorted상태로 유지되어야 한다.
⇨ 사용된 자료구조: Binary Search Tree, Sorted Doubly Linked list, Queue

(4) Delete cast

마찬가지로 movie의 title key를 이용하여 BST에서 검색을 진행해준다. 검색 결과 pre-order tree iterator가 반환된다. 그리고 iterator에 data가 존재한다면 castlist에 삭제하고자 하는 배우의 name이 존재하는지도 확인해준다. 이 역시 존재함이 확인되면 list에서 cast name을 제거해주되, sorted order가 유지된다.
⇨ 사용된 자료구조: Binary Search Tree, Sorted Doubly Linked list, Queue

(5) Find movie

movie의 title key를 이용하여 BST에서 검색을 진행해준다. 검색 결과 pre-order tree iterator가 반환된다. 그리고 iterator에 data가 있다면 traverse된 tree의 current node에 있는 data를 반환해준다. 그리고 이 정보가 사용자가 입력한 정보와 같은지 비교한다.

⇨ 사용된 자료구조: Binary Search Tree, Sorted Doubly Linked list, Queue

(6) Find movie actor

새로운 in-order iterator를 생성하고, 이로써 root로부터 in-order tree iterator가 반환된다. iterator가 data를 가지고 있다면 list에 사용자가 찾고자 하는 배우의 name이 존재하는지 확인한다. 그리고 list에 배우의 name도 존재한다면 List<Movie>타입의 parameter인 result에 영화정보를 삽입하고(배우가 여러 영화에 출연했을 수 있으므로) result를 return 해준다.

⇨ 사용된 자료구조: Binary Search Tree, Sorted Doubly Linked list, Queue

(7) Find movie rating

새로운 in-order iterator를 생성하고, 이로써 root로부터 in-order tree iterator가 반환된다. iterator가 data를 가지고 있다면 이에 대한 rating을 반환받고, 사용자가 입력한 rating의 값과 비교한다. 일치한다면 이번에도 result에 그 정보를 저장해주고, while문을 나가면 result값을 반환해준다.

⇨ 사용된 자료구조: Binary Search Tree, Sorted Doubly Linked list, Queue

(8) Find movie year

새로운 in-order iterator를 생성하고, 이로써 root로부터 in-order tree iterator가 반환된다. iterator가 data를 가지고 있다면 이에 대한 year를 반환받고, 사용자가 입력한 year의 값과 비교한다. 일치하면 result에 그 정보를 저장해준다. 그리고 while 문을 나오면 result의 정보가 반환된다.

⇨ 사용된 자료구조: Binary Search Tree, Sorted Doubly Linked list, Queue

(9) Find movie period

새로운 in-order iterator를 생성하고, 이로써 root로부터 in-order tree iterator가 반환된다. iterator가 data를 가지고 있다면 이에 대한 year를 반환받는다. 여기서는 사용자가 두 개의 year(year1, year2)에 대한 정보를 입력하게 되는데, year1은 반환받은 year의 값보다 작거나 같고, year2는 반환받은 year의 값보다 크거나 같아야 삽입이 이루어진다. 역시나 삽입은 result에서 이루어지며 while문을 나가면 result가 반환된다.

⇨ 사용된 자료구조: Binary Search Tree, Sorted Doubly Linked list, Queue

(10) print movies order
in-order iterator를 통해 Binary Search Tree에 있는 data에 순차적으로 접근하여 출력해준다.
⇨ 사용된 자료구조: Binary Search Tree, Queue

Q1) 다루는 자료의 삽입, 삭제 패턴은 어떤 것 인가? 리스트, 스택, 큐 등 중에서 어느 것을 사용할까?
⇨ 다루는 자료 중 List의 경우 삽입과 삭제 시에도 순서대로 유지되며, Tree의 경우 recursion을 통해 tree를 traverse하며 삽입 및 삭제를 해준다./ List와 Queue가 활용되었다.

Q2) 사용하려는 자료구조는 삽입, 삭제, 검색 중 어느 것에 효율적인가?
⇨ Doubly Linked List는 검색보다는 삽입 및 삭제 시에 유리한 자료구조이다. 특히 수정이나 삭제가 빈번하게 이루어져야 하는 경우에는 BST가 아닌 Sorted Linked List를 이용하였다.

Q3) 프로젝트의 환경은 동적인 환경(삽입/삭제가 빈번하게 발생) 또는 정적인 환경인가? 여기에 적합한 자료구조를 사용하는가?
⇨ 동적인 환경
'Movie Database Application'은 위에서도 언급했듯 데이터의 삽입과 삭제가 빈번하게 발생하는 동적인 환경이다. 따라서 이에 적합한 Linked list 자료구조를 사용하였다.

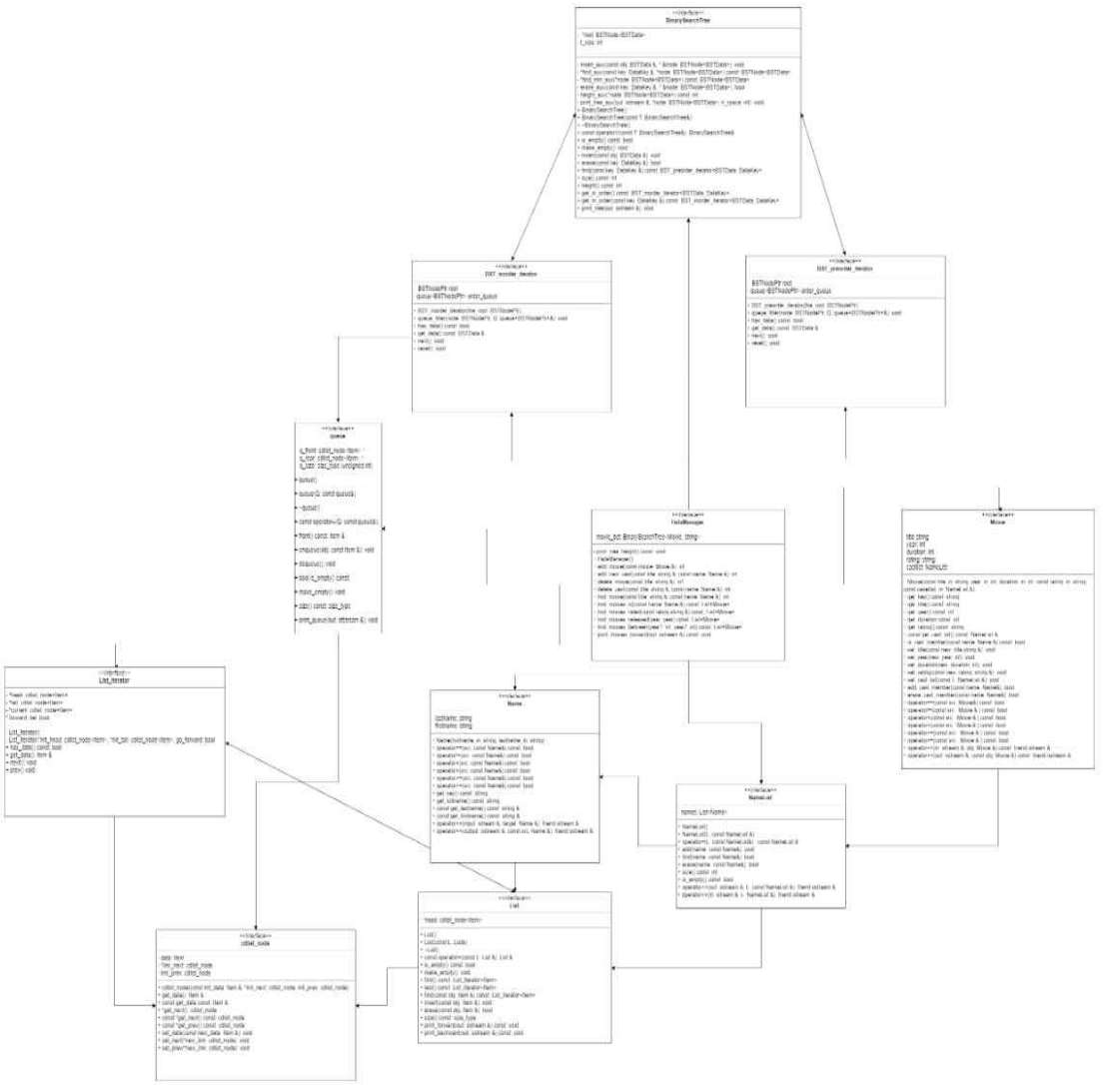
Q4) 리스트가 적합할까? 이진탐색트리가 적합할까?
⇨ 이진탐색트리
이진탐색트리는 이진탐색(Binary Search)와 연결리스트(Linked list)를 결합한 자료구조의 일종이다. 이진탐색트리의 경우 균형이 잡혀있는 한, 각 항목에 대한 검색경로가 연결리스트 보다 훨씬 짧다.(삽입, 삭제, 검색의 평균 시간 복잡도: $LogN$)
따라서 효율적인 탐색이 가능할 뿐만 아니라 자료의 빈번한 삽입과 삭제도 가능하다는 장점을 가진다.
'Movie Database Application'의 경우 영화이름, 배우이름, 등급, 개봉년도, 상영시간에 대한 탐색기능을 제공한다. 따라서 연결리스트보다는 이진탐색트리가 적합하다고 판단하였다.

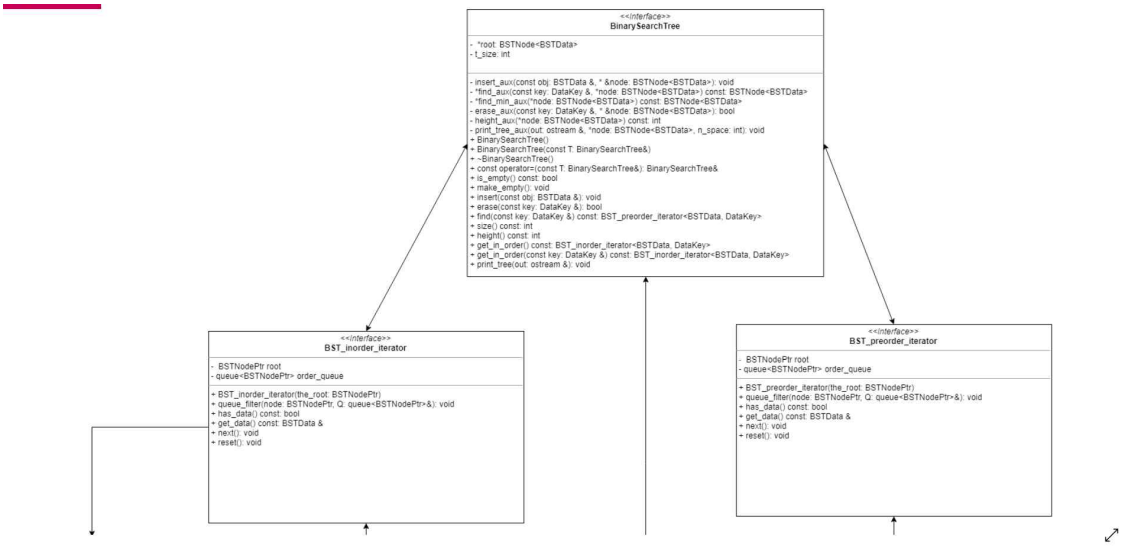
Q5) 배열을 사용한 리스트가 적합할 까? 연결 구조를 사용한 리스트가 적합할까?
⇨ 연결구조
배열의 경우 index를 안다면 데이터 접근속도가 굉장히 빠르다는 장점이 있다. 그

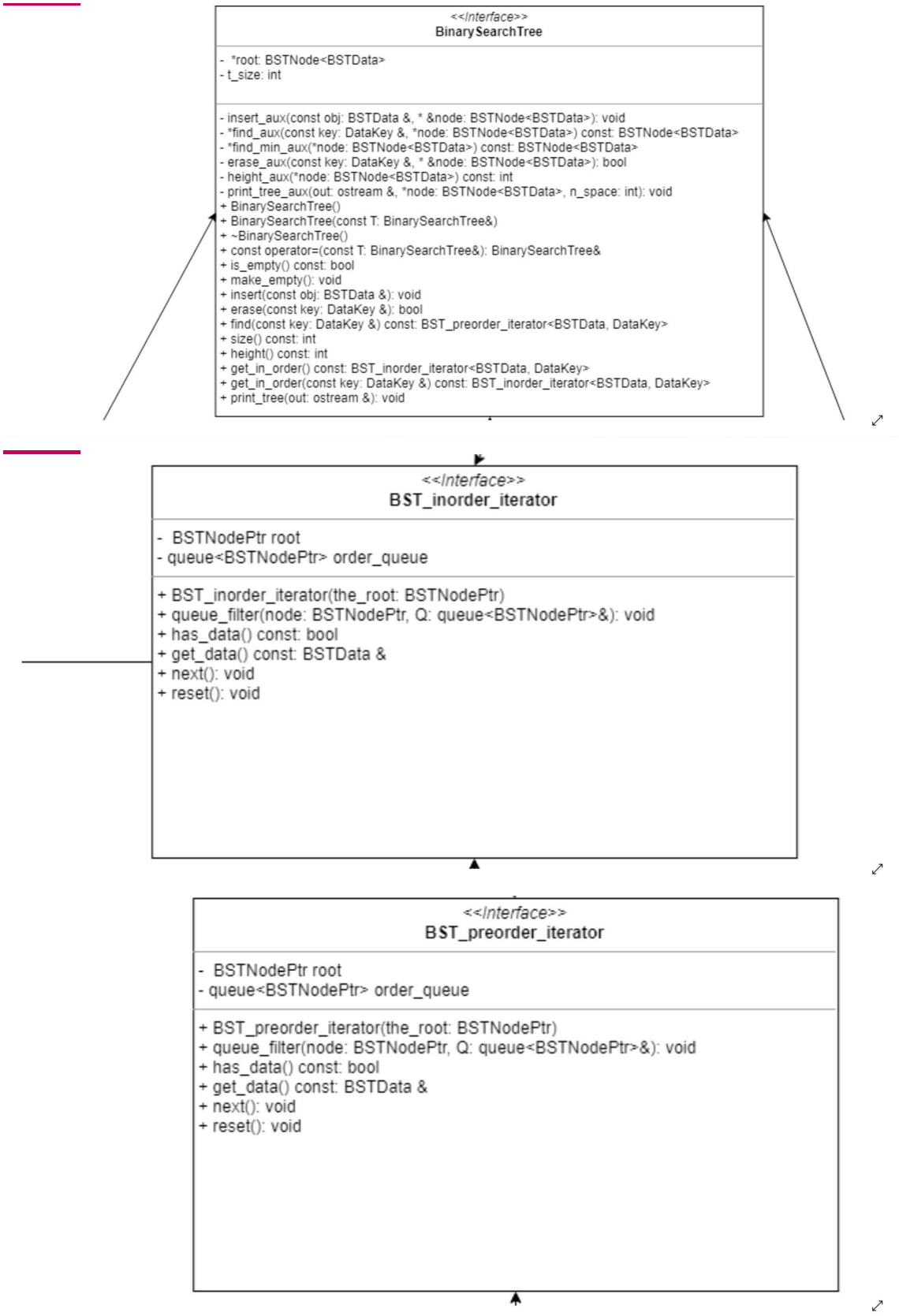
러나 데이터의 삽입이나 삭제 시, 삽입 및 삭제가 이루어진 위치의 다음부터 모든 데이터의 위치를 변경해야 한다는 단점이 있다.
반면에 연결리스트의 경우 데이터에 대한 접근속도는 배열보다 느리다. 그러나 데이터의 삽입 및 삭제 시에 주소만 바꾸어 주면 되기 때문에 배열보다 훨씬 용이하게 쓸 수 있다.

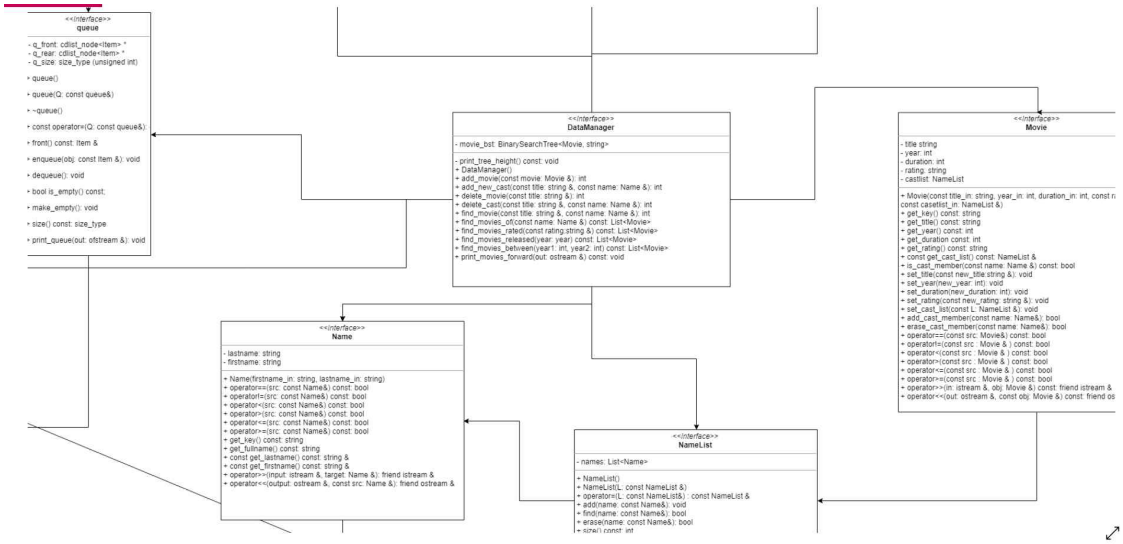
Q6) Linked list가 적합할까? Doubly Linked list가 적합할까?
⇨ Doubly linked list
일반적인 Linked list는 다음 노드의 탐색만 가능하지만, Doubly linked list는 두 개의 포인터가 있어 앞뒤로 탐색이 가능하다. 따라서 일반적인 Linked list에 비해 효율적인 탐색이 가능하고, 최대 탐색시간을 반으로 줄일 수 있다.
그러나 linked list와 달리 이전 노드를 지정하기 위한 변수가 하나 더 필요하여 메모리를 더 많이 사용하며, 구현이 조금 더 복잡해진다는 단점이 있다.
따라서 데이터양이 많고, 데이터 접근이 빈번하게 이루어지며, 삽입 삭제가 거의 없는 경우는 배열을 사용하고, 데이터양이 많지 않고, 데이터의 삽입 및 삭제가 빈번하게 이루어질 경우에는 연결리스트를 사용하는 것이 좋다.
'Movie Database Application'의 경우 영화이름의 삽입, 삭제, 영화배우 이름의 삽입, 삭제 등의 기능을 통해 삽입과 삭제가 빈번하게 발생하기 때문에 연결리스트가 적합하다고 판단하였다.

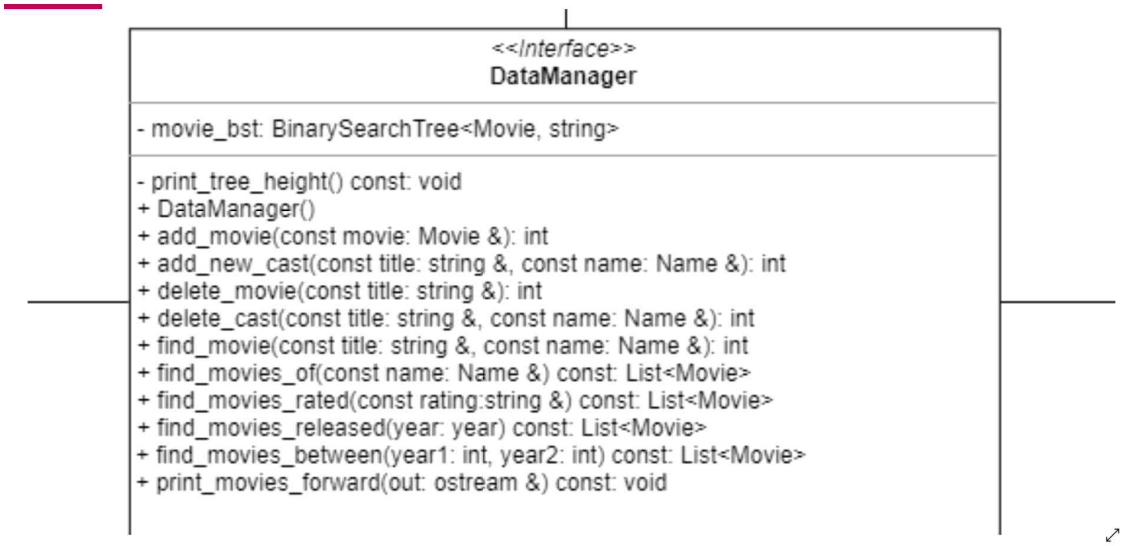
- 클래스 다이어그램 (png파일을 첨부하였습니다.)

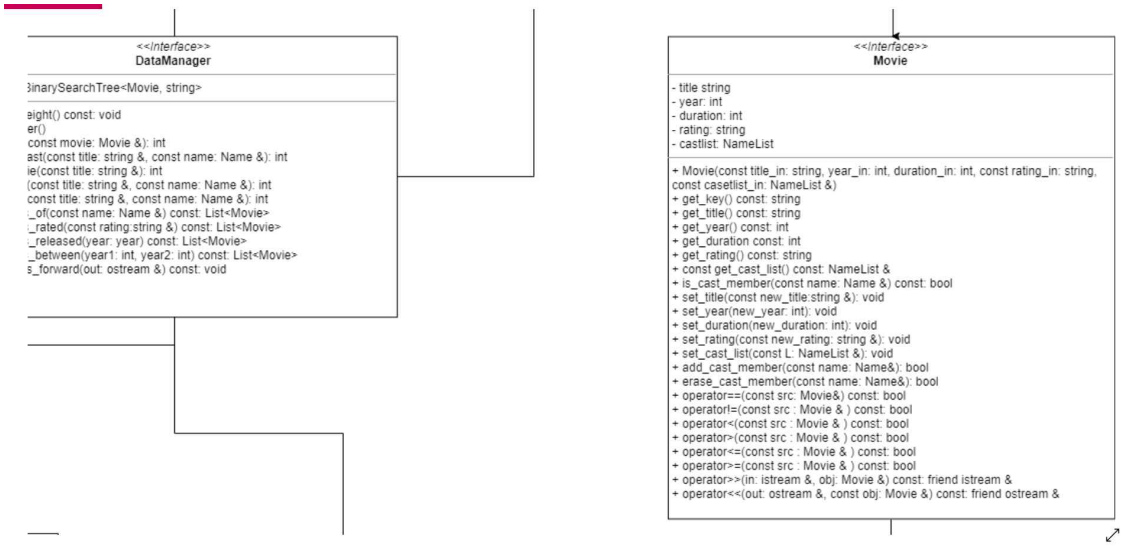


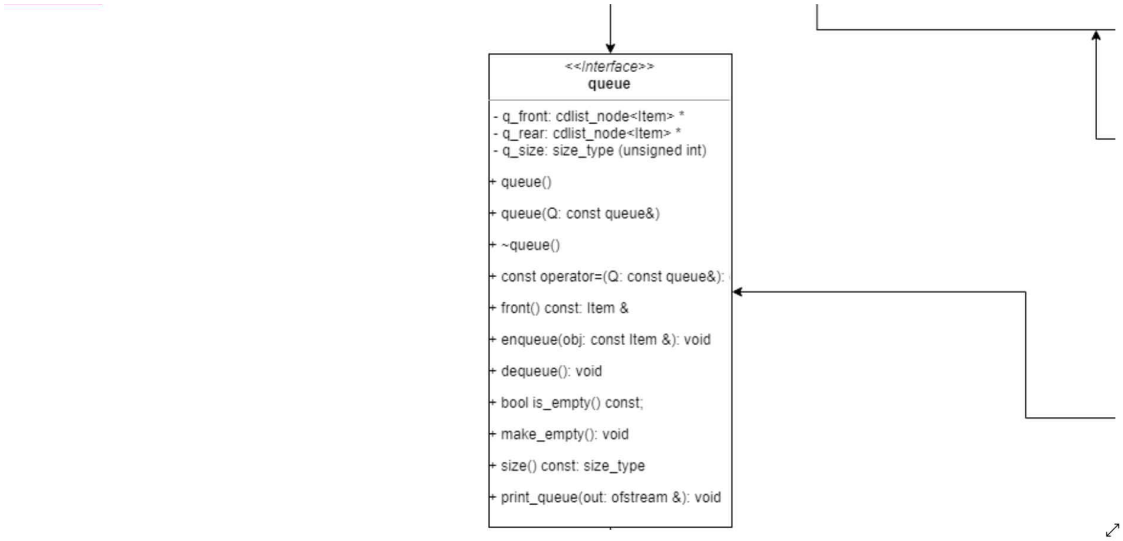


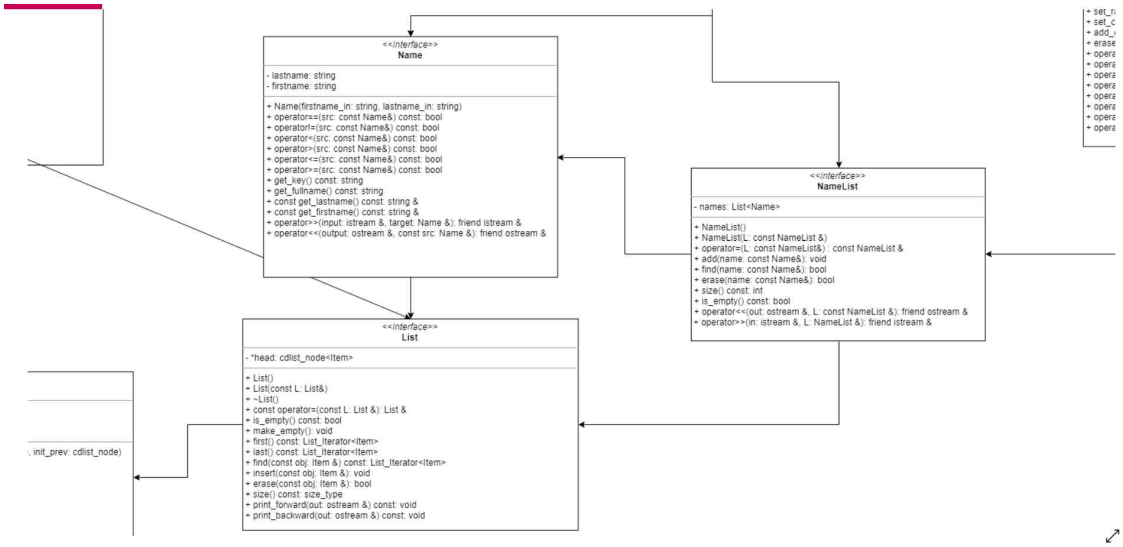

<<Interface>>
BinarySearchTree
- *root: BSTNode<BSTData>
- t_size: int
- insert_aux(const obj: BSTData &, * &node: BSTNode<BSTData>): void
- *find_aux(const key: DataKey &, *node: BSTNode<BSTData>) const: BSTNode<BSTData>
- *find_min_aux(*node: BSTNode<BSTData>) const: BSTNode<BSTData>
- erase_aux(const key: DataKey &, * &node: BSTNode<BSTData>): bool
- height_aux(*node: BSTNode<BSTData>) const: int
- print_tree_aux(out: ostream &, *node: BSTNode<BSTData>, n_space: int): void
+ BinarySearchTree()
+ BinarySearchTree(const T: BinarySearchTree&)
+ ~BinarySearchTree()
+ const operator=(const T: BinarySearchTree&): BinarySearchTree&
+ is_empty() const: bool
+ make_empty(): void
+ insert(const obj: BSTData &): void
+ erase(const key: DataKey &): bool
+ find(const key: DataKey &) const: BST_preorder_iterator<BSTData, DataKey>
+ size() const: int
+ height() const: int
+ get_in_order() const: BST_inorder_iterator<BSTData, DataKey>
+ get_in_order(const key: DataKey &) const: BST_inorder_iterator<BSTData, DataKey>
+ print_tree(out: ostream &): void
<<Interface>>
BST_inorder_iterator
- BSTNodePtr root
- queue<BSTNodePtr> order_queue
+ BST_inorder_iterator(the_root: BSTNodePtr)
+ queue_filter(node: BSTNodePtr, Q: queue<BSTNodePtr>&): void
+ has_data() const: bool
+ get_data() const: BSTData &
+ next(): void
+ reset(): void
<<Interface>>
BST_preorder_iterator
- BSTNodePtr root
- queue<BSTNodePtr> order_queue
+ BST_preorder_iterator(the_root: BSTNodePtr)
+ queue_filter(node: BSTNodePtr, Q: queue<BSTNodePtr>&): void
+ has_data() const: bool
+ get_data() const: BSTData &
+ next(): void
+ reset(): void

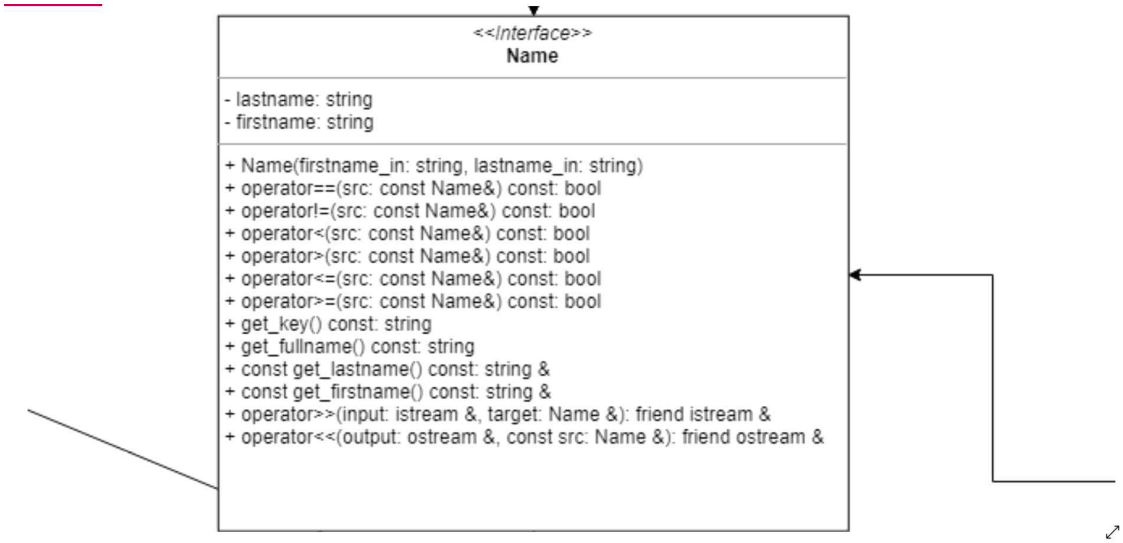

<<Interface>>
queue
- q_front: cdlist_node<Item> *
- q_rear: cdlist_node<Item> *
- q_size: size_type (unsigned int)
+ queue()
+ queue(Q: const queue&)
+ ~queue()
+ const operator=(Q: const queue&):
+ front() const: Item &
+ enqueue(obj: const Item &): void
+ dequeue(): void
+ bool is_empty() const;
+ make_empty(): void
+ size() const: size_type
+ print_queue(out: ofstream &): void
<<Interface>>
DataManager
- movie_bst: BinarySearchTree<Movie, string>
- print_tree_height() const: void
+ DataManager()
+ add_movie(const movie: Movie &): int
+ add_new_cast(const title: string &, const name: Name &): int
+ delete_movie(const title: string &): int
+ delete_cast(const title: string &, const name: Name &): int
+ find_movie(const title: string &, const name: Name &): int
+ find_movies_of(const name: Name &) const: List<Movie>
+ find_movies_rated(const rating:string &) const: List<Movie>
+ find_movies_released(year: year) const: List<Movie>
+ find_movies_between(year1: int, year2: int) const: List<Movie>
+ print_movies_forward(out: ostream &) const: void
<<Interface>>
Movie
- title string
- year: int
- duration: int
- rating: string
- castlist: NameList
+ Movie(const title_in: string, year_in: int, duration_in: int, const r
const casetlist_in: NameList &)
+ get_key() const: string
+ get_title() const: string
+ get_year() const: int
+ get_duration const: int
+ get_rating() const: string
+ const get_cast_list() const: NameList &
+ is_cast_member(const name: Name &) const: bool
+ set_title(const new_title:string &): void
+ set_year(new_year: int): void
+ set_duration(new_duration: int): void
+ set_rating(const new_rating: string &): void
+ set_cast_list(const L: NameList &): void
+ add_cast_member(const name: Name&): bool
+ erase_cast_member(const name: Name&): bool
+ operator==(const src: Movie&) const: bool
+ operator!=(const src : Movie &) const: bool
+ operator<(const src : Movie &) const: bool
+ operator>(const src : Movie &) const: bool
+ operator<=(const src : Movie &) const: bool
+ operator>=(const src : Movie &) const: bool
+ operator>>(in: istream &, obj: Movie &) const: friend istream &
+ operator<<(out: ostream &, const obj: Movie &) const: friend os
<<Interface>>
Name
- lastname: string
- firstname: string
+ Name(firstname_in: string, lastname_in: string)
+ operator==(src: const Name&) const: bool
+ operator!=(src: const Name&) const: bool
+ operator<(src: const Name&) const: bool
+ operator>(src: const Name&) const: bool
+ operator<=(src: const Name&) const: bool
+ operator>=(src: const Name&) const: bool
+ get_key() const: string
+ get_fullname() const: string
+ const get_lastname() const: string &
+ const get_firstname() const: string &
+ operator>>(input: istream &, target: Name &): friend istream &
+ operator<<(output: ostream &, const src: Name &): friend ostream &
<<Interface>>
NameList
- names: List<Name>
+ NameList()
+ NameList(L: const NameList &)
+ operator=(L: const NameList&) : const NameList &
+ add(name: const Name&): void
+ find(name: const Name&): bool
+ erase(name: const Name&): bool
+ size() const: int

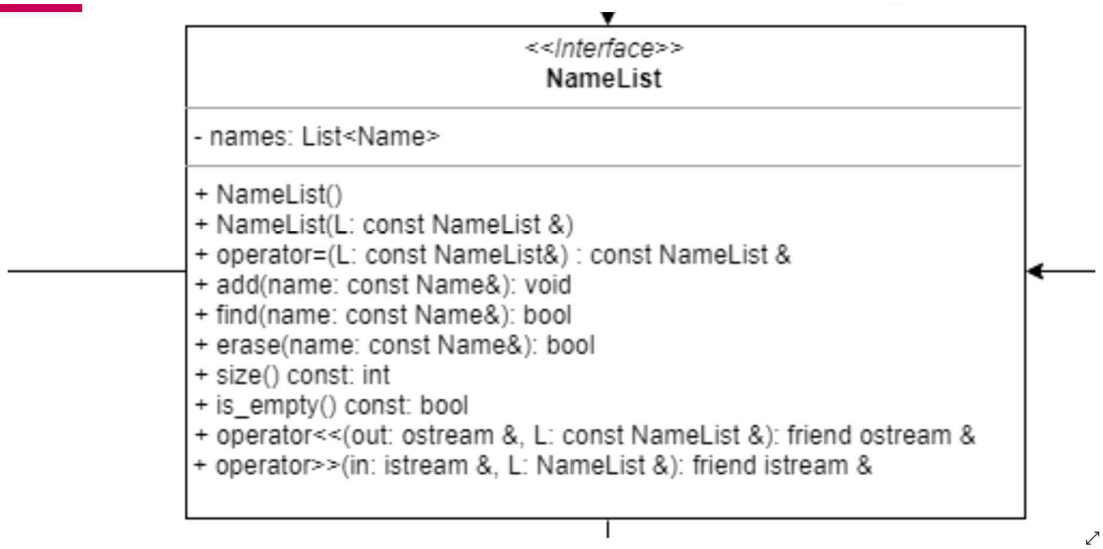

<<Interface>>
DataManager
- movie_bst: BinarySearchTree<Movie, string>
- print_tree_height() const: void
+ DataManager()
+ add_movie(const movie: Movie &): int
+ add_new_cast(const title: string &, const name: Name &): int
+ delete_movie(const title: string &): int
+ delete_cast(const title: string &, const name: Name &): int
+ find_movie(const title: string &, const name: Name &): int
+ find_movies_of(const name: Name &) const: List<Movie>
+ find_movies_rated(const rating:string &) const: List<Movie>
+ find_movies_released(year: year) const: List<Movie>
+ find_movies_between(year1: int, year2: int) const: List<Movie>
+ print_movies_forward(out: ostream &) const: void

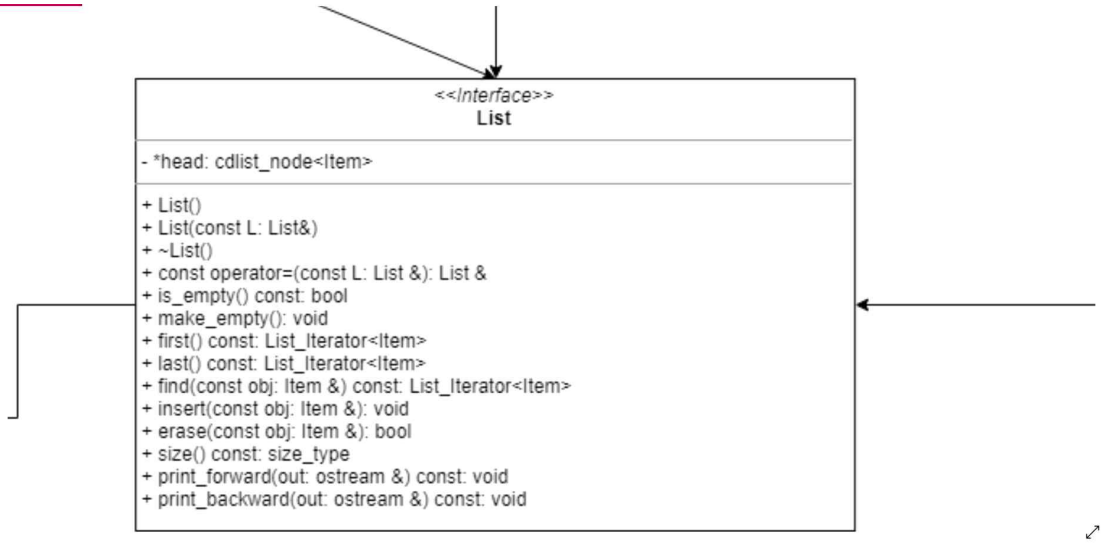

<<Interface>>
DataManager
<<Interface>>
Movie
- title string
- year: int
- duration: int
- rating: string
- castlist: NameList
+ Movie(const title_in: string, year_in: int, duration_in: int, const rating_in: string, const casetlist_in: NameList &)
+ get_key() const: string
+ get_title() const: string
+ get_year() const: int
+ get_duration const: int
+ get_rating() const: string
+ const get_cast_list() const: NameList &
+ is_cast_member(const name: Name &) const: bool
+ set_title(const new_title:string &): void
+ set_year(new_year: int): void
+ set_duration(new_duration: int): void
+ set_rating(const new_rating: string &): void
+ set_cast_list(const L: NameList &): void
+ add_cast_member(const name: Name&): bool
+ erase_cast_member(const name: Name&): bool
+ operator==(const src: Movie&) const: bool
+ operator!=(const src : Movie &) const: bool
+ operator<(const src : Movie &) const: bool
+ operator>(const src : Movie &) const: bool
+ operator<=(const src : Movie &) const: bool
+ operator>=(const src : Movie &) const: bool
+ operator>>(in: istream &, obj: Movie &) const: friend istream &
+ operator<<(out: ostream &, const obj: Movie &) const: friend ostream &

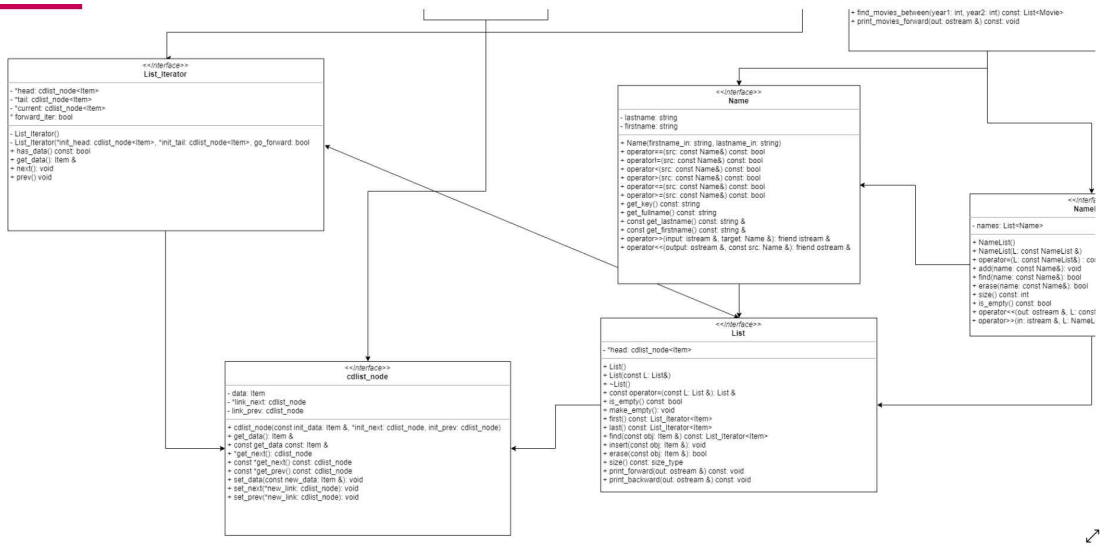

<<Interface>>
queue
- q_front: cdlist_node<Item> *
- q_rear: cdlist_node<Item> *
- q_size: size_type (unsigned int)
+ queue()
+ queue(Q: const queue&)
+ ~queue()
+ const operator=(Q: const queue&):
+ front() const: Item &
+ enqueue(obj: const Item &): void
+ dequeue(): void
+ bool is_empty() const;
+ make_empty(): void
+ size() const: size_type
+ print_queue(out: ofstream &): void

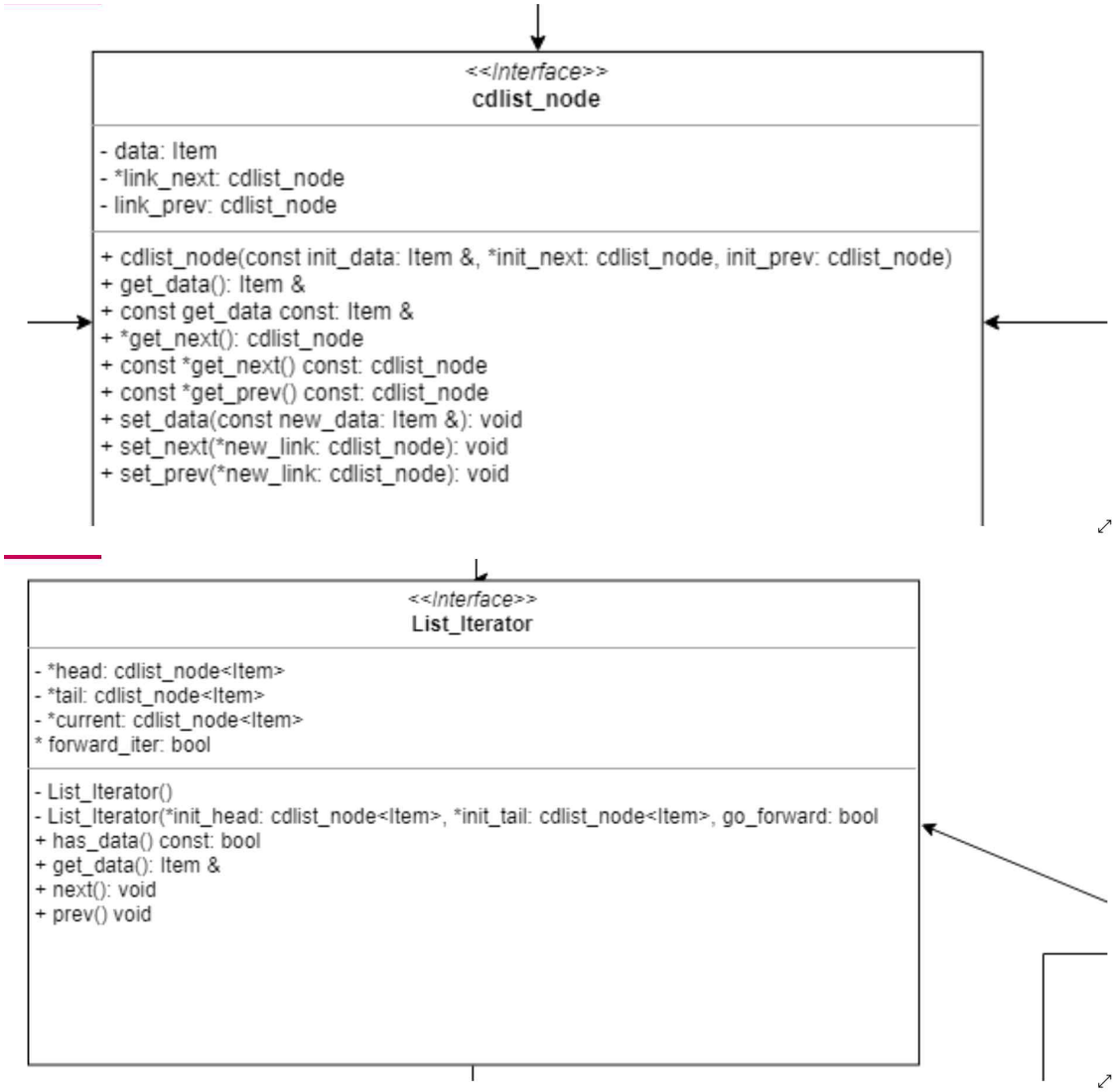

<<Interface>>
Name
- lastname: string
- firstname: string
+ Name(firstname_in: string, lastname_in: string)
+ operator==(src: const Name&) const: bool
+ operator!=(src: const Name&) const: bool
+ operator<(src: const Name&) const: bool
+ operator>(src: const Name&) const: bool
+ operator<=(src: const Name&) const: bool
+ operator>=(src: const Name&) const: bool
+ get_key() const: string
+ get_fullname() const: string
+ const get_lastname() const: string &
+ const get_firstname() const: string &
+ operator>>(input: istream &, target: Name &): friend istream &
+ operator<<(output: ostream &, const src: Name &): friend ostream &
<<Interface>>
NameList
- names: List<Name>
+ NameList()
+ NameList(L: const NameList &)
+ operator=(L: const NameList&) : const NameList &
+ add(name: const Name&): void
+ find(name: const Name&): bool
+ erase(name: const Name&): bool
+ size() const: int
+ is_empty() const: bool
+ operator<<(out: ostream &, L: const NameList &): friend ostream &
+ operator>>(in: istream &, L: NameList &): friend istream &
<<Interface>>
List
- *head: cdlist_node<Item>
+ List()
+ List(const L: List&)
+ ~List()
+ const operator=(const L: List &): List &
+ is_empty() const: bool
+ make_empty(): void
+ first() const: List_Iterator<Item>
+ last() const: List_Iterator<Item>
+ find(const obj: Item &) const: List_Iterator<Item>
+ insert(const obj: Item &): void
+ erase(const obj: Item &): bool
+ size() const: size_type
+ print_forward(out: ostream &) const: void
+ print_backward(out: ostream &) const: void


<<Interface>>
Name
- lastname: string
- firstname: string
+ Name(firstname_in: string, lastname_in: string)
+ operator==(src: const Name&) const: bool
+ operator!=(src: const Name&) const: bool
+ operator<(src: const Name&) const: bool
+ operator>(src: const Name&) const: bool
+ operator<=(src: const Name&) const: bool
+ operator>=(src: const Name&) const: bool
+ get_key() const: string
+ get_fullname() const: string
+ const get_lastname() const: string &
+ const get_firstname() const: string &
+ operator>>(input: istream &, target: Name &): friend istream &
+ operator<<(output: ostream &, const src: Name &): friend ostream &


<<Interface>>
NameList
- names: List<Name>
+ NameList()
+ NameList(L: const NameList &)
+ operator=(L: const NameList&) : const NameList &
+ add(name: const Name&): void
+ find(name: const Name&): bool
+ erase(name: const Name&): bool
+ size() const: int
+ is_empty() const: bool
+ operator<<(out: ostream &, L: const NameList &): friend ostream &
+ operator>>(in: istream &, L: NameList &): friend istream &

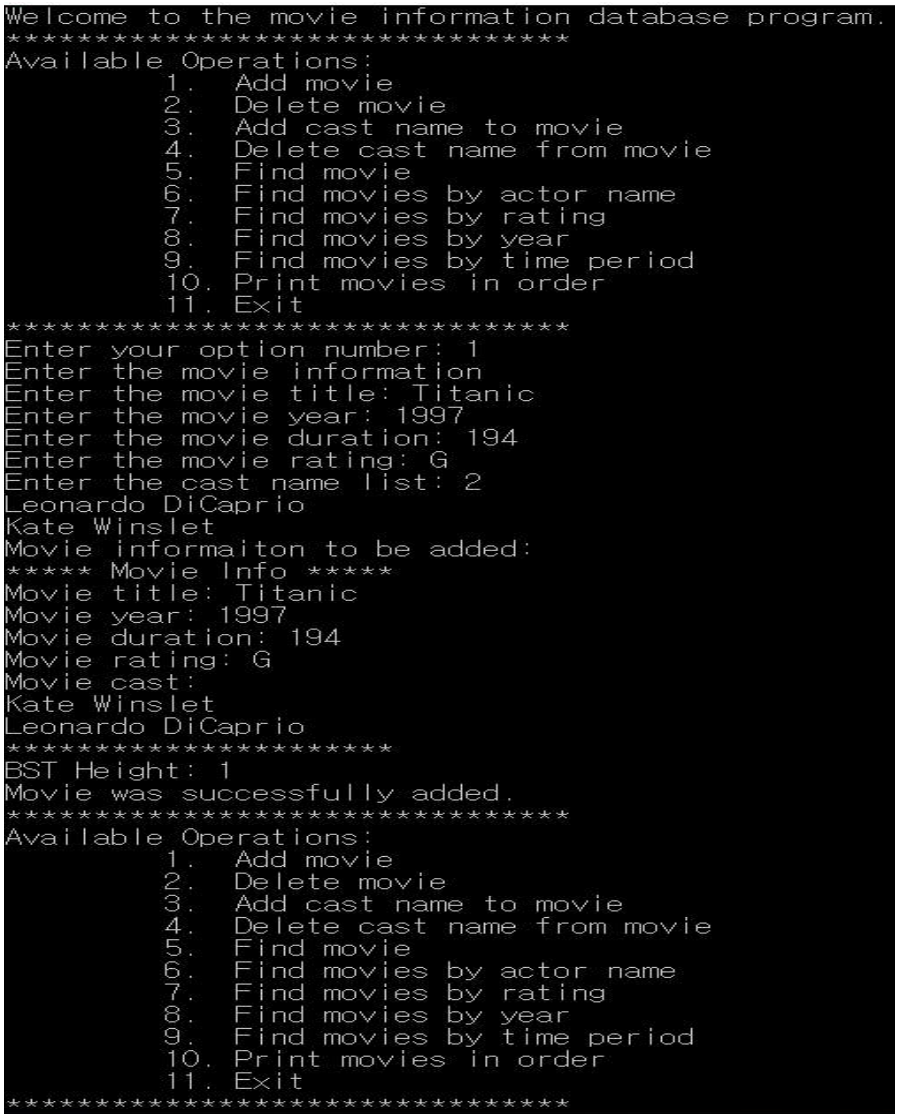

<<Interface>>
List
- *head: cdlist_node<Item>
+ List()
+ List(const L: List&)
+ ~List()
+ const operator=(const L: List &): List &
+ is_empty() const: bool
+ make_empty(): void
+ first() const: List_Iterator<Item>
+ last() const: List_Iterator<Item>
+ find(const obj: Item &) const: List_Iterator<Item>
+ insert(const obj: Item &): void
+ erase(const obj: Item &): bool
+ size() const: size_type
+ print_forward(out: ostream &) const: void
+ print_backward(out: ostream &) const: void


<<Interface>>
List_Iterator
<<Interface>>
Name
<<Interface>>
List
<<Interface>>
cdlist_node


<<Interface>>
cdlist_node
- data: Item
- *link_next: cdlist_node
- link_prev: cdlist_node
+ cdlist_node(const init_data: Item &, *init_next: cdlist_node, init_prev: cdlist_node)
+ get_data(): Item &
+ const get_data const: Item &
+ *get_next(): cdlist_node
+ const *get_next() const: cdlist_node
+ const *get_prev() const: cdlist_node
+ set_data(const new_data: Item &): void
+ set_next(*new_link: cdlist_node): void
+ set_prev(*new_link: cdlist_node): void
<<Interface>>
List_Iterator
- *head: cdlist_node<Item>
- *tail: cdlist_node<Item>
- *current: cdlist_node<Item>
* forward_iter: bool
- List_Iterator()
- List_Iterator(*init_head: cdlist_node<Item>, *init_tail: cdlist_node<Item>, go_forward: bool
+ has_data() const: bool
+ get_data(): Item &
+ next(): void
+ prev() void

3. 시작품 계획

최종적으로 상품화가 된다면 아마 'Application'형태로 출시가 될 것이다. 그러나 시작품 단계에서는 interface보다는 프로그램 자체의 정확도나 다양한 자료구조를 적절히 활용하는 데에 초점을 두고자한다. 그리고 현재는 콘솔 창을 통해 프로그램을 이용할 수 있는 단계에 있다.

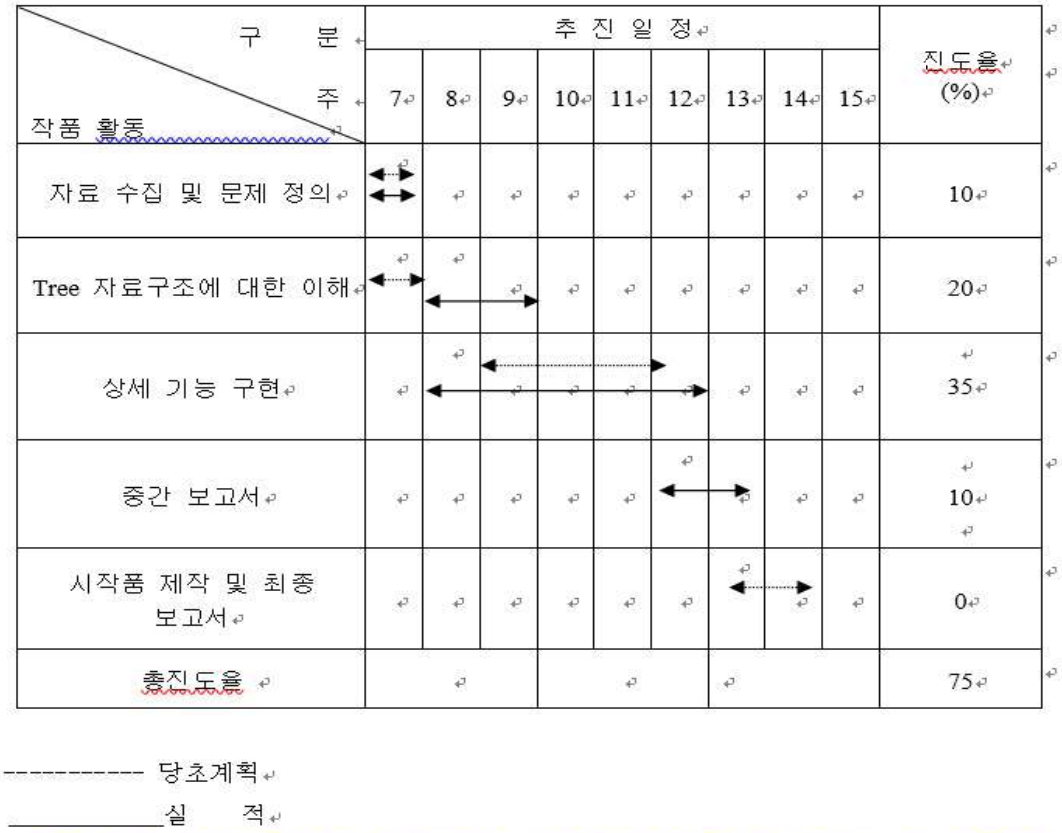


앞으로 프로그램에 대한 코드가 완벽히 구현된다면 이를 Application이 아닌 웹 페이지를 통해 구현해보고자 한다. 물론 application형태로 까지 구현해본다면 더욱 좋았겠지만 현재의 나로서는 앱 프로그래밍에 대한 경험이 거의 없다. 따라서 현실적으로 내가 가진 지식을 활용하여 간단한 웹페이지를 만들어서 사용자를 위한 interface를 구현해보고자 한다.

4. 작품 평가 항목

평가 항목	목표	가중치 (%)	평가 기준안
자료구조	3개 이상의 자료구조(적어도 하나이상의 고급 자료구조)가 적절히 활용되었는가?	30	목표에 100% 이상 도달 시 100점 목표에 80% 도달 시 80점
질의응답 시간	질의 후 답변이 30초 이내에 나와야 한다.	30	
블록도, 다이어그램	블록도, 다이어그램을 적절히 활용하여 보고서를 작성했는가?	20	
알고리즘	자신의 프로그램이 어떻게 구현되는지 잘 이해하고 있는가?	20	

5. 추진 결과 보고 (중간)



구분 / 주 / 작품 활동	추진 일정 7	8	9	10	11	12	13	14	15	진도율 (%)
자료 수집 및 문제 정의	←	→								10
Tree 자료구조에 대한 이해	←	→	→							20
상세 기능 구현		→	→	→	→	→				35
중간 보고서						→	→			10
시작품 제작 및 최종 보고서							←	→		0
총진도율										75

---------- 당초계획

_____실 적

6. 작품의 독창성

영화예매, 영화 다시보기와 같은 앱이나, 서비스는 이미 많이 존재한다. 그러나 아직 자신이 본 영화를 체계적으로 저장해둘 수 있는 서비스는 잘 없다. 기존의 영화 관련 프로그램은 영화를 예매하고, 자신이 예매해서 본 영화의 기록을 확인하는 것과 같은 서비스를 제공해왔다. 좀 더 나아가서는 블로그 등을 활용하여 자신이 본 영화에 대한 줄거리, 평 등을 기록할 수 있었다. 그러나 전자의 경우 자신만의 영화 목록을 관리하기엔 다소 부족하다고 생각하며, 후자의 경우에는 바쁜 현대인에게 다소 부담스러울 수 있는 작업이라 생각한다. 따라서 나는 기존의 프로그램들과 달리 이름, 영화배우 등의 간단한 내용 기입만으로도 자신이 본 영화의 목록을 관리할 수 있을 뿐만 아니라, 누구나 쉽고 빠르게 자신의 영화목록을 관리할 수 있는 서비스를 만들었다.

덧붙여서 나의 경우 영화를 좋아하는 사람으로서 내가 감상한 영화에 대한 '기록'은 굉장히 의미 있는 일 중 하나이다. 따라서 나처럼 영화를 좋아하는 사용자들에게는 이러한 서비스가 일기만큼 의미 있는 기록을 관리해줄 수 있을 거라 생각한다. 이 application을 통해 사람들이 간단하게나마 자신만의 기록을 꾸준히 해나가며 소소한 행복도 느낄 수 있으면 좋겠다.

7. 전문가 자문

자문을 구하지 않았다.

8. 예상 소요 예산

무료로 사용가능한 software만을 이용하여 구현해 보고자한다.

* 주의 사항

- 프로젝트 주제가 창의적인 내용을 포함할 것
- 자료구조 3개 이상 사용
- 트리, 그래프 등의 고급 자료구조 하나 이상 사용
- 사용된 각 자료구조가 무엇을 하기 위해 사용하는지 기술
- 꼭 필요한 경우가 아니면 하나의 자료를 여러 개의 자료에 중복해서 저장되는 것을 피할 것
- 자료구조의 원래 용도, 장단점을 파악하여 적절한 자료구조가 선택되어야 함
  - 예 1: 다루는 자료의 삽입/삭제의 패턴은 어떤 것인가? 리스트, 스택, 큐 등 중에서 어느 것을 사용할까?
  - 예 2: 사용하려는 자료구조는 삽입, 삭제, 검색 중 어는 것에 효율적인가?
  - 예 3: 프로젝트의 환경은 동적인 환경(삽입/삭제가 빈번하게 발생) 또는 정적인 환경인가? 여기에 적합한 자료구조를 사용하는가?
  - 예 4: 리스트가 적합할까? 이진탐색트리가 적합할까?
  - 예 5: 배열을 사용한 리스트가 적합할까? 연결구조를 사용한 리스트가 적합할까?