Sequence Class Template
Documentation

0. **Author**

Name: Jan Radzimiński

Field of study: Computer Science, Faculty of Electronics and Information Technology

Index number: 293052

Group: 102

1. **General Information**

Sequence project was implemented and compiled using Code Blocks and MinGW compiler, with C++11 standard.

Sequence is a class template constructed as singly linked list. List consists of ‘’Nodes’’ which are elements of structure declared within the class. All struct elements consist of template type Key which works as ‘’ID’’ of the element, and Info which stores data in given list element. Class supports all basic operation on the list, such as insertion and deleting of elements at front, back or after key, printing list, operators, data access etc. (all functions are described later in document). Class also has iterator implemented as class within class Sequence. Iterator was especially useful while implementing non-class function Split – function witch takes sequence and with specified parameters splits it into two other sequences. Whole class implementation was placed in file „Sequence.h’’ and Split function in „Split.h”.

2. **Sequence Class Members**

Private:

Member:	**Explanation:**

struct Node:
Key ID;
Info Data;
Node *next;
	Nodes are simply elements of the list consisting of template type ‘Key ID’ which works like ID of the node, ‘Info Data’ to store data in the element, and pointer ‘next’ to next element of the list.

Node *head;	Pointer head is simply the pointer to the first element of the list. If list is empty, it’s just a nullpointer.

int list_size;	Integer consisting of number of elements in the list, controlled by all add/remove methods. If list is empty it is equal to 0.

Public:

Member:	**Explanation:**
class Iterator;	Iterator class, friend to the Sequence class, which is used to navigate through list by functions outside of the class. It allows to 'hide' all Node pointer values of the class so user using it does not have access to memory part of the class. More about this class can be found later in the document.

3. Sequence Class Functions

Public:

Function:	**Explanation:**
Constructors:	
Sequence<Key, Info>( ):	Empty constructor of the class, which assigns head pointer to null and list size to 0.
~Sequence<Key, Info>( );	Destructor of the class, which consist of Clear_List() method, which deletes all the nodes of the class and assures that there are no memory leaks after destruction of it.
Sequence<Key, Info>(const Sequence<Key, Info>& source);	Copy constructor of the class, which firstly set its pointer to null (for clear_list() to work properly) and then uses assignment operator to copy source list to itself.
Operators:	
Sequence<Key, Info>& operator=(const Sequence<Key, Info>& source);	Assignment operator, which firstly clears (this) list to assure its empty, and then goes through Nodes of source list and adds copy of each Node to 'this', using Push_Back method.
Sequence<Key, Info> operator+(const Sequence<Key, Info>& source);	Add operator, which adds all Nodes from source list after last node of 'this' list, using Push_Back method.

Sequence<Key, Info>& operator+=(const Sequence<Key, Info>& source)	*this = *this + source;
bool operator==(const Sequence<Key, Info>& source) const;	Boolean operator which returns true only if two lists have exact length and all nodes in both lists are identical and in the same order.
bool operator!=(const Sequence<Key, Info>& source) const	return !(*this==source);
bool operator>(const Sequence<Key, Info>& source) const	Operator which returns true if 'this' list size is bigger then source's.
bool operator<(const Sequence<Key, Info>& source) const	Opposite as previous operator.
friend std::ostream& operator<<(std::ostream& os, const Sequence<K, I>& seq);	Operator << which allows to print whole list in different streams.
Node Add Functions:	
bool **Push_Front**(const Key &ID, const Info &Data);	Function which adds Node with given Key and Info values at the beginning of the list. It directs head into new added element and next of it to previous first element. Remark: function is bool type just because during tests I would know if function returned true (if program exited it). Generally it can return only true.
bool **Push_Back**(const Key &ID, const Info &Data);	Similar to previous one, but this one add new Node at the end of the list – if list was empty at beginning it points head into it.
bool **Add_At_Position** (const Key &ID, const Info &Data, const int &position);	Function which adds given node at given position (counting from 0) and pushes rest of nodes. If given position is wrong (negative, or bigger than list size) it returns false.
bool **Add_After_Key** (const Key &ID, const Info &Data, const Key &after_key, int where=1);	Function which adds new Node after Node with key same as 'after_key', and which appeared where'th time. If where is not specified it default value is 1 (first appearance of given key).
bool **Add_Before_Key** (const Key &ID, const Info &Data, const Key &bef_key, int where=1);	Same as previous one, but adds node before given key.

Node Remove Functions:	
bool **Pop_Front**();	Function that deletes first element of the list, and points head at second element. If list is empty it returns false.
bool **Pop_Back**();	Function that deletes last element of the list, and points second-last element to null. If list is empty it returns false.
int **Remove_By_Key** (const Key &rem_key);	Function that removes all Nodes with same key as rem_key.
bool **Remove_At_Position** (const int &position);	Function that removes Node at given position on the list (if given position is wrong it returns false).
void **Clear_List**();	Function that goes through every Node in the list and deletes it. At the end it sets head to null and list size to 0. Used in the destructor of class.
List Info:	
bool **Is_Empty**() const	Returns true if list_size is not 0 and false otherwise.
int **List_Length**() const	Returns list_size.
Data Access:	
Info **Get_Info**(const Key &nkey, int where=1) const;	Goes through a list and returns data member of a where-th appearance of a node with key same as nkey. If such node weren't found it returns 0.
Info **Front_Node_Info**() const;	Returns Data of first Node and 0 if list Is empty.
Info **Back_Node_Info**() const;	Returns Data of last Node and 0 if list Is empty.
Key **Front_Key**() const;	Returns Key of First Node and 0 if list Is empty.
Key **Back_Key**() const;	Returns Key of Last Node and 0 if list is empty.
Displaying the List:	
void Print_List() const;	Prints (cout) whole list in larger scale than operator <<.
void Print_Front() const	Prints (cout) first node of list, of nothing if list is empty.

void Print_Back() const	Prints (cout) last node of list, of nothing if list is empty.
Print_By_Key(const Key &k, int where=1) const	Prints (cout) where-th node with same key as k or nothing if such node doesn’t exist.
Iterators	
Iterator ibegin() const	Returns Iterator pointing to first Node or null if list is empty.
const Iterator const_ibegin() const	Returns const Iterator pointing to first Node or null if list is empty.
Iterator iend() const	Returns iterator pointing to null.
const Iterator const_iend() const	Returns const iterator pointing to null.

Private:

//Remark: these are the functions which take Node pointers as arguments, used by public methods of class but not visible to user of the class.

Function:	**Explanation:**
bool Push_Front(Node *new_node);	Same as previous push front but adds already created node. If newnode points to null return false.
bool Push_Back(Node *new_node);	Same as first in this table.
bool Add_At_Position (Node *new_node, const int &position);	Same as first in this table.
bool Add_After_Key (Node *new_node, const Key &after_key, int where=1);	Same as first in this table.
void Print_Node(Node *curr) const;	Same as first in this table.
void Print_Node(Node *curr, int num) const;	Used during function testing also prints number of given node.

4. Iterator Class Members & Functions

Members (private):

Member:	**Explanation:**
Node *current;	‘Hided’ pointer to element of the list.

Functions:

Function:	**Explanation:**
Iterator();	Empty constructor, sets current to null.
Iterator(Node *ptr):	**Private** constructor which sets current to ptr. (available for Sequence class because its friend of iterator class).
~Iterator()	Destructor of class.
Iterator(const Iterator& other)	Copy constructor sets this->currant as same as other.current .
Iterator& operator=(const Iterator& other)	Assignment operator sets this->currant as same as other.current .
bool operator==(const Iterator& source) const	Returns true if this and source current pointers are the same.
Iterator operator++(int)	Navigator of iterator, moves to next element in the list (if its possible – current is not null).
Key Show_Key() const	Returns key of node that current points to or 0 if currents is null.
Info Show_Data() const	Returns Info of node that current points to or 0 if currents is null.

5. Tests

Tests for this class template were written in main function. They consist of test of every function included in point 3 (which are every function of a class) + function split.