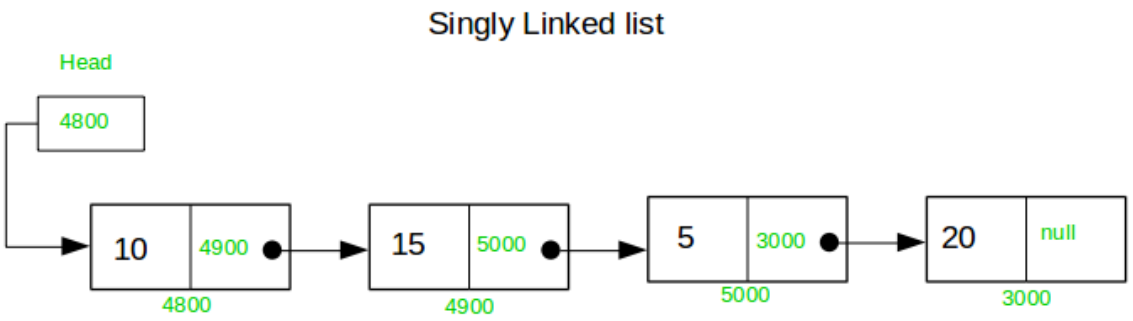
EADS-Lab 1

Single linked list



Student: Xián García Nogueira K-6077
Lecturer: Turlej Dariusz

19/04/2021

Index:

Description:

This project consists of a single linked list implemented in c++ for structure data with a key and the information for each node of the structure.
In the source files are the **Sequence.ccp** and **Sequence.h** that are full Single linked list implemented, they must be included in the in the desired code with:
“#include "Sequence.cpp"
and the file must be in the project folder.
Then in the main.cpp are the tests and the 2 externals functions merge_key and merge_pos

For initialization a new Single list this is the code:

Sequence<Key, Info> name;

Aim of the Lab:

The aim of the Lab is to implement a full SingleLinkedList with 2 external functions which one merge 2 lists with the starting position and other with the starting key.
Then testing all the methods.

Methods:

Sequence();

Constructor for the list

~Sequence();

Deconstructor that delete the list

Sequence(const Sequence<Key, Info>& copy);

Copy constructor, that create a Sequence initializing it with a deep copy of another sequence

Sequence<Key, Info> &operator=(const Sequence<Key, Info> & move);

Assign a copy of a sequence to an existing sequence using the prev constructor.

Sequence<Key, Info> &operator-=(const Sequence<Key, Info> & newList);

Subtract to the current Sequence the sequence given by parameters

Sequence<Key, Info> &operator+=(const Sequence<Key, Info> & newList);

Add to the current Sequence the sequence given by parameters

friend Sequence<key, info> operator-(const Sequence<key, info>& substractOp2, const Sequence<key, info>& substractOp);

Subtract to the first Sequence given by parameters with the second, this constructor uses the constructor “operator-=”

friend Sequence<key, info> operator+(const Sequence<key, info>& addOp2, const Sequence<key, info>& addOp);

Add to the first Sequence given by parameters with the second, this constructor uses the constructor “operator+=”

bool insertAfter(const Key &newKey, const Info &newInfo, const Key &where, int occurance);

Insert new node to the list after the key given by parameter and in which occurrence it should be in case the keys are repeated, if the keys are not repeated just occurance=1
return true if all go bood and false if there is an error on the request.

bool insertBefore(const Key &newKey, const Info &newInfo, const Key &where, int occurance);

Insert new node to the list before the key given by parameter and in which occurrence it should be in case the keys are repeated, if the keys are not repeated just occurrence=1
return true if all go bood and false if there is an error on the request.

bool insertAtEnd(const Key &newKey, const Info &newInfo);

Insert a new Node with the values by parameters at the end of the list
return true if all go bood and false if there is an error on the request.

bool insertAtBeg(const Key &newKey, const Info &newInfo);

Insert a new Node with the values by parameters at the beginning of the list
return true if all go good and false is there is an error on the request.

bool isEmpty();

Returns true if the current sequence is empty or false if not

bool itsNodeEmpty(int index);

Returns true if the current sequence in the position given by parameter is pointed to a nullptr and false if not

Key getKey(int index);

Returns the key from the current sequence in the position given by parameter.

Info getInfo(int index);

Returns the info from the current sequence in the position given by parameter.

bool deleteElement(const Key &delKey, int occurance);

Delete the node with the key given by parameter and in which occurrence it should be in case the keys are repeated, if the keys are not repeated just occurrence=1
return true if all go good and false is there is an error on the request.

void prinf();

Print in console the actual list

bool search(const Key &what, int occurance);

Search if exists in the node given by parameters in the actual list.
Return false if not, true if yes.

bool AssertEquals(string wantedOutput);

METHOD ONLY FOR TEST-DEVEOLOPMENT

Method only created for testing, for comparing the data from the current Sequence given in the format that I created to know if the values from the sequence are the same of the desired ones given by parameters.

Testing:

For testing i use an auxiliary class **main.cpp** where with the method **AssertEquals** from the linked List I tested all the methods.
For execute it you only need to run the main.cpp

Constructors tests:

Code:

```
cout << endl
     << "--------Constructors---------" << endl;
Sequence<int, int> test1Constructors;
test1Constructors.insertAtBeg(2, 5);
test1Constructors.insertAtBeg(6, 7);

Sequence<int, int> test2Constructors = test1Constructors;

test1Constructors.insertAtBeg(1, 1);
test2Constructors.insertAtBeg(3, 3);
cout << "Test 0.1: " << (test2Constructors.AssertEquals("[3,3][6,7][2,5]") == 1 ? "true" : "false") << endl;
test2Constructors.prinf();



Sequence<int, int> test4Constructors;
Sequence<int, int> test5Constructors = test4Constructors;//call to copy constructor
cout << "Test 0.2(empty copyconstructor): " << (test5Constructors.AssertEquals("[]") == 1 ? "true" : "false") << endl;

Sequence<int, int> test6Constructors;
test6Constructors.insertAtBeg(2, 5);
test6Constructors.insertAtBeg(6, 7);
Sequence<int, int> test3Constructors;
test3Constructors.insertAtEnd(3,6);
test3Constructors= test6Constructors;// calls to assignment constructor
cout << "Test 0.3: " << (test3Constructors.AssertEquals("[6,7][2,5]") == 1 ? "true" : "false") << endl;
test3Constructors.prinf();


Sequence<int, int> test7Constructors;
test7Constructors+= test3Constructors;
cout << "Test 0.4(+=operator, empty first List): " << (test7Constructors.AssertEquals("[6,7][2,5]") == 1 ? "true" : "false") << endl;
test7Constructors.prinf();
```

Output:

```
--------Constructors---------
Test 0.1: true
[3,3][6,7][2,5]
Test 0.2(empty copyconstructor): true
Test 0.3: true
[6,7][2,5]
Test 0.4(+=operator, empty first List): true
[6,7][2,5]
```

Code:

```
test7Constructors+=test3Constructors;
cout << "Test 0.5(+=operator): " << (test7Constructors.AssertEquals("[6,7][2,5][6,7][2,5]") == 1 ? "true" : "false") << endl;
test7Constructors.prinf();

Sequence<int, int> test8Constructors;
test7Constructors+= test8Constructors;
cout << "Test 0.6(+=operator, empty second List): " << (test7Constructors.AssertEquals("[6,7][2,5][6,7][2,5]") == 1 ? "true" : "false") << endl;
test7Constructors.prinf();

Sequence<int, int> test9Constructors;
test9Constructors=test7Constructors + test8Constructors;
cout << "Test 0.7(+operator, empty second List): " << (test9Constructors.AssertEquals("[6,7][2,5][6,7][2,5]") == 1 ? "true" : "false") << endl;
test9Constructors.prinf();

Sequence<int, int> test10Constructors;
test10Constructors = test7Constructors + test9Constructors;
cout << "Test 0.8(+operator): " << (test10Constructors.AssertEquals("[6,7][2,5][6,7][2,5][6,7][2,5][6,7][2,5]") == 1 ? "true" : "false") << endl;
test10Constructors.prinf();
```

Output:

```
Test 0.5(+=operator): true
[6,7][2,5][6,7][2,5]
Test 0.6(+=operator, empty second List): true
[6,7][2,5][6,7][2,5]
Test 0.7(+operator, empty second List): true
[6,7][2,5][6,7][2,5]
Test 0.8(+operator): true
[6,7][2,5][6,7][2,5][6,7][2,5][6,7][2,5]
```

Code:

```
Sequence<int, int> test11Constructors;
Sequence<int, int> test12Constructors;

test11Constructors-= test12Constructors;
cout << "Test 0.9(-=operator, empty first List): " << (test12Constructors.AssertEquals("[]") == 1 ? "true" : "false") << endl;
test12Constructors.prinf();


test11Constructors.insertAtBeg(1,1);
test11Constructors.insertAtBeg(2,2);
test11Constructors-= test12Constructors;
cout << "Test 0.10(-=operator, empty second List): " << (test11Constructors.AssertEquals("[2,2][1,1]") == 1 ? "true" : "false") << endl;
test11Constructors.prinf();


test12Constructors.insertAtBeg(1,1);
test11Constructors-= test12Constructors;
cout << "Test 0.11(-=operator): " << (test11Constructors.AssertEquals("[2,2]") == 1 ? "true" : "false") << endl;
test11Constructors.prinf();
```

Output:

```
Test 0.9(-=operator, empty first List): true
[]
Test 0.10(-=operator, empty second List): true
[2,2][1,1]
Test 0.11(-=operator): true
[2,2]
```

Code:

```
Sequence<int, int> test13Constructors;
Sequence<int, int> test14Constructors;
Sequence<int, int> test15Constructors;
test15Constructors.insertAtEnd(5,5);
test13Constructors=test14Constructors - test15Constructors;
cout << "Test 0.12(-operator, empty first List): " << (test13Constructors.AssertEquals("[]") == 1 ? "true" : "false") << endl;
test13Constructors.prinf();


test13Constructors=test15Constructors - test14Constructors;
cout << "Test 0.13(-operator, empty second List): " << (test13Constructors.AssertEquals("[5,5]") == 1 ? "true" : "false") << endl;
test13Constructors.prinf();
```

Output:

```
Test 0.12(-operator, empty first List): true
[]
Test 0.13(-operator, empty second List): true
[5,5]
```

Code:

```
 test14Constructors.insertAtEnd(0,0);
 test14Constructors.insertAtEnd(1,1);
 test14Constructors.insertAtEnd(2,5);
 test14Constructors.insertAtEnd(3,5);
 test14Constructors.insertAtEnd(3,3);
 test14Constructors.insertAtEnd(4,5);

 test15Constructors.insertAtEnd(3,0);
 test15Constructors.insertAtEnd(0,5);
 test15Constructors.insertAtEnd(6,5);
 test15Constructors.insertAtEnd(0,5);
 test13Constructors=test14Constructors - test15Constructors;
 cout << "Test 0.14(-operator): " << (test13Constructors.AssertEquals("[1,1][2,5][3,3][4,5]") == 1 ? "true" : "false") << endl;
 test13Constructors.prinf();
```

Output:

```
Test 0.14(-operator): true
[1,1][2,5][3,3][4,5]
```

Insert at beginning tests:

Code:

```
cout<< endl << "--------INSERT AT BEG---------" << endl;
Sequence<int, int> test1InsertBeg;
test1InsertBeg.insertAtBeg(2, 5);
test1InsertBeg.insertAtBeg(6, 7);
test1InsertBeg.insertAtBeg(7, 9);
test1InsertBeg.insertAtBeg(1, 2);

cout << "Test 1: " << (test1InsertBeg.AssertEquals("[1,2][7,9][6,7][2,5]") == 1 ? "true" : "false") << endl;
test1InsertBeg.prinf();
```

Output:

```
--------INSERT AT BEG---------
Test 1: true
[1,2][7,9][6,7][2,5]
```

Insert at end tests:

Code:

```
cout<< endl << "--------INSERT AT END---------" << endl;
Sequence<int, int> test1InsertAtEnd;
test1InsertAtEnd.insertAtEnd(2, 5);
test1InsertAtEnd.insertAtEnd(6, 7);
test1InsertAtEnd.insertAtEnd(7, 9);
test1InsertAtEnd.insertAtEnd(1, 2);

cout << "Test 2: " << (test1InsertAtEnd.AssertEquals("[2,5][6,7][7,9][1,2]") == 1 ? "true" : "false") << endl;
test1InsertAtEnd.prinf();
```

Output:

```
--------INSERT AT END---------
Test 2: true
[2,5][6,7][7,9][1,2]
```

Delete and Search tests:

Code:

```
cout<< endl << "--------DELETE AND SEARCH---------" << endl;

Sequence<int, int> test1Delete;

cout <<  "Test 3.1: " << (test1Delete.deleteElement(2, 5) == false ? "true" : "false") << endl;

test1Delete.insertAtBeg(2, 5);
test1Delete.insertAtBeg(6, 7);
test1Delete.insertAtBeg(7, 9);
test1Delete.insertAtBeg(7, 45);
test1Delete.insertAtBeg(7, 10);
test1Delete.insertAtBeg(1, 2);
test1Delete.deleteElement(7, 1);

cout << "Test 3.2: " << (test1Delete.AssertEquals("[1,2][7,45][7,9][6,7][2,5]") == 1 ? "true" : "false") << endl;
test1Delete.prinf();

test1Delete.deleteElement(7, 2);
cout << "Test 3.3: " << (test1Delete.AssertEquals("[1,2][7,45][6,7][2,5]") == 1 ? "true" : "false") << endl;
test1Delete.prinf();

test1Delete.deleteElement(3, 2);
cout << "Test 3.4: " << (test1Delete.AssertEquals("[1,2][7,45][6,7][2,5]") == 1 ? "true" : "false") << endl;


test1Delete.deleteElement(7, 2);
cout << "Test 3.5: " << (test1Delete.AssertEquals("[1,2][7,45][6,7][2,5]") == 1 ? "true" : "false") << endl;

test1Delete.deleteElement(7, 0);
cout << "Test 3.6: " << (test1Delete.AssertEquals("[1,2][7,45][6,7][2,5]") == 1 ? "true" : "false") << endl;
```

Output:

```
--------DELETE AND SEARCH---------
Test 3.1: true
Test 3.2: true
[1,2][7,45][7,9][6,7][2,5]
Test 3.3: true
[1,2][7,45][6,7][2,5]
Test 3.4: true
Test 3.5: true
Test 3.6: true
```

Insert after tests:

Code:

```
cout << endl<< "--------INSERT AFTER---------" << endl;

Sequence<int, int> test1InsertAfter;

 cout << "Test 4.1: " << (test1InsertAfter.insertAfter(8, 9,2,1) == false ? "true" : "false") << endl;
test1InsertAfter.insertAtBeg(2, 5);
test1InsertAfter.insertAtBeg(6, 7);
test1InsertAfter.insertAfter(8, 9,2,1);
test1InsertAfter.insertAfter(8, 10,6,1);
test1InsertAfter.insertAfter(1, 1,8,2);


cout << "Test 4.2: " << (test1InsertAfter.AssertEquals("[6,7][8,10][2,5][8,9][1,1]") == 1 ? "true" : "false") << endl;
test1InsertAfter.prinf();

test1InsertAfter.insertAfter(1, 1,8,5);
cout << "Test 4.2: " << (test1InsertAfter.AssertEquals("[6,7][8,10][2,5][8,9][1,1]") == 1 ? "true" : "false") << endl;


test1InsertAfter.insertAfter(1, 1,11,5);
cout << "Test 4.2: " << (test1InsertAfter.AssertEquals("[6,7][8,10][2,5][8,9][1,1]") == 1 ? "true" : "false") << endl;
```

Output:

```
--------INSERT AFTER---------
Test 4.1: true
Test 4.2: true
[6,7][8,10][2,5][8,9][1,1]
Test 4.2: true
Test 4.2: true
```

Insert before tests:

Code:

```
    cout<< endl << "--------INSERT BEFORE---------" << endl;

Sequence<int, int> test1InsertBefore;

 cout << "Test 5.1: " << (test1InsertBefore.insertBefore(8, 9,2,1) == false ? "true" : "false") << endl;
test1InsertBefore.insertAtBeg(2, 5);
test1InsertBefore.insertAtBeg(6, 7);
test1InsertBefore.insertBefore(8, 9,2,1);
test1InsertBefore.insertBefore(8, 10,6,1);
test1InsertBefore.insertBefore(1, 1,8,2);


cout << "Test 5.2: " << (test1InsertBefore.AssertEquals("[8,10][6,7][1,1][8,9][2,5]") == 1 ? "true" : "false") << endl;
test1InsertBefore.prinf();

test1InsertBefore.insertBefore(1, 1,8,5);
cout << "Test 5.3: " << (test1InsertBefore.AssertEquals("[8,10][6,7][1,1][8,9][2,5]") == 1 ? "true" : "false") << endl;


test1InsertBefore.insertBefore(1, 1,11,5);
cout << "Test 5.4: " << (test1InsertBefore.AssertEquals("[8,10][6,7][1,1][8,9][2,5]") == 1 ? "true" : "false") << endl;
```

Output:

```
--------INSERT BEFORE---------
Test 5.1: true
Test 5.2: true
[8,10][6,7][1,1][8,9][2,5]
Test 5.3: true
Test 5.4: true
```

Merge by position tests:

Code:

```
cout<< endl << "--------MERGE BY POSITION---------" << endl;

Sequence<int, int> test1MergePos;
Sequence<int, int> test1MergePos2;



test1MergePos.insertAtEnd(0, 0);
test1MergePos.insertAtEnd(1, 1);
test1MergePos.insertAtEnd(2, 2);
test1MergePos.insertAtEnd(3, 3);
test1MergePos.insertAtEnd(4, 4);
test1MergePos.insertAtEnd(5, 5);
test1MergePos.insertAtEnd(6, 6);
test1MergePos.insertAtEnd(7, 7);
test1MergePos.insertAtEnd(8, 8);
test1MergePos.insertAtEnd(9, 9);


test1MergePos2.insertAtEnd(0, 0);
test1MergePos2.insertAtEnd(1, 10);
test1MergePos2.insertAtEnd(2, 20);
test1MergePos2.insertAtEnd(3, 30);
test1MergePos2.insertAtEnd(4, 40);
test1MergePos2.insertAtEnd(5, 50);
test1MergePos2.insertAtEnd(6, 60);
test1MergePos2.insertAtEnd(7, 70);
test1MergePos2.insertAtEnd(8, 80);
test1MergePos2.insertAtEnd(9, 90);
```

```
Sequence<int, int> test1MergePosOut=merge_pos(test1MergePos,3,1,test1MergePos2,2,2,4);
cout << "Test 6.2: " << (test1MergePosOut.AssertEquals("[3,3][2,20][3,30][4,4][4,40][5,50][5,5][6,60][7,70][6,6][8,80][9,90]") == 1 ? "true" : "false") << endl;
test1MergePosOut.prinf();

Sequence<int, int> test1MergePosOut2= merge_pos(test1MergePos,3,1,test1MergePos2,2,2,0);
cout << "Test 6.3(0count): " << (test1MergePosOut2.AssertEquals("[]") == 1 ? "true" : "false") << endl;
test1MergePosOut2.prinf();

Sequence<int, int> test1MergePosOut3= merge_pos(test1MergePos,3,1,test1MergePos2,2,2,20);
cout << "Test 6.4(bigcount): " << (test1MergePosOut3.AssertEquals("[3,3][2,20][3,30][4,4][4,40][5,50][5,5][6,60][7,70][6,6][8,80][9,90][7,7][8,8][9,9]") == 1 ? "true" : "false") << endl;
test1MergePosOut3.prinf();

Sequence<int, int> test1MergePosOut4= merge_pos(test1MergePos,3,0,test1MergePos2,2,0,8);
cout << "Test 6.5(0length): " << (test1MergePosOut4.AssertEquals("[]") == 1 ? "true" : "false") << endl;
test1MergePosOut4.prinf();

Sequence<int, int> test1MergePosOut5= merge_pos(test1MergePos,20,1,test1MergePos2,20,2,8);
cout << "Test 6.6(bigstartpos): " << (test1MergePosOut5.AssertEquals("[]") == 1 ? "true" : "false") << endl;
test1MergePosOut5.prinf();


Sequence<int, int> test1MergePosOut6= merge_pos(test1MergePos,0,1,test1MergePos2,0,2,10);
cout << "Test 6.7:(0startpos) " << (test1MergePosOut6.AssertEquals("[0,0][0,0][1,10][1,1][2,20][3,30][2,2][4,40][5,50][3,3][6,60][7,70][4,4][8,80][9,90][5,5][6,6][7,7][8,8][9,9]") == 1 ? "true" : "false") << endl;
test1MergePosOut6.prinf();

Sequence<int, int> test1Mergetest;
Sequence<int, int> test1Mergetest2;

Sequence<int, int> test1MergePosOut7= merge_pos(test1Mergetest,0,1,test1Mergetest2,0,2,10);
cout << "Test 6.8:(nullsequences) " << (test1MergePosOut7.AssertEquals("[]") == 1 ? "true" : "false") << endl;
test1MergePosOut7.prinf();

Sequence<int, int> test1MergePosOut8= merge_pos(test1Mergetest,-5,-4,test1Mergetest2,-5,-2,-10);
cout << "Test 6.9:(negatives) " << (test1MergePosOut8.AssertEquals("[]") == 1 ? "true" : "false") << endl;
test1MergePosOut8.prinf();

Sequence<int, int> test1MergePosOut9= merge_pos(test1MergePos,4,1,test1MergePos2,-1,-2,3);
cout << "Test 6.10(onenegative):  " << (test1MergePosOut9.AssertEquals("[4,4][5,5][6,6]") == 1 ? "true" : "false") << endl;
test1MergePosOut9.prinf();
```

Output:

```
--------MERGE BY POSITION---------
Test 6.2: true
[3,3][2,20][3,30][4,4][4,40][5,50][5,5][6,60][7,70][6,6][8,80][9,90]
Test 6.3(0count): true
[]
Test 6.4(bigcount): true
[3,3][2,20][3,30][4,4][4,40][5,50][5,5][6,60][7,70][6,6][8,80][9,90][7,7][8,8][9,9]
Test 6.5(0length): true
[]
Test 6.6(bigstartpos): true
[]
Test 6.7:(0startpos) true
[0,0][0,0][1,10][1,1][2,20][3,30][2,2][4,40][5,50][3,3][6,60][7,70][4,4][8,80][9,90][5,5][6,6][7,7][8,8][9,9]
Test 6.8:(nullsequences) true
[]
Test 6.9:(negatives) true
[]
Test 6.10(onenegative):  true
[4,4][5,5][6,6]
```

Merge by key tests:

Code:

```
cout << endl
     << "--------MERGE BY KEY---------" << endl;

test1MergePos.insertAtEnd(3, 50);

Sequence<int, int> test1MergeKeyOut2 = merge_key(test1MergePos, 3, 2, 1, test1MergePos2, 4, 1, 2, 2);
cout << "Test 7.1: " << (test1MergeKeyOut2.AssertEquals("[3,50][4,40][5,50][6,60][7,70]") == 1 ? "true" : "false") << endl;
test1MergeKeyOut2.prinf();

Sequence<int, int> test1MergeKeyOut3 = merge_key(test1MergePos, 3, 2, 20, test1MergePos2, 4, 1, 20, 2);
cout << "Test 7.2(biggerLength):  " << (test1MergeKeyOut3.AssertEquals("[3,50][4,40][5,50][6,60][7,70][8,80][9,90]") == 1 ? "true" : "false") << endl;
test1MergeKeyOut3.prinf();

Sequence<int, int> test1MergeKeyOut4 = merge_key(test1MergePos, 3, 2, 0, test1MergePos2, 4, 1, 0, 2);
cout << "Test 7.3(0length):  " << (test1MergeKeyOut4.AssertEquals("[]") == 1 ? "true" : "false") << endl;
test1MergeKeyOut4.prinf();

Sequence<int, int> test1MergeKeyOut5 = merge_key(test1MergePos, 3, 2, 1, test1MergePos2, 4, 1, 2, 0);
cout << "Test 7.4(0count):  " << (test1MergeKeyOut5.AssertEquals("[]") == 1 ? "true" : "false") << endl;
test1MergeKeyOut5.prinf();

Sequence<int, int> test1MergeKeyOut6 = merge_key(test1MergePos, 3, 2, 1, test1MergePos2, 4, 1, 2, 1);
cout << "Test 7.5(1count):  " << (test1MergeKeyOut6.AssertEquals("[3,50][4,40][5,50]") == 1 ? "true" : "false") << endl;
test1MergeKeyOut6.prinf();

Sequence<int, int> test1MergeKeyOut7 = merge_key(test1MergePos, 3, 0, 1, test1MergePos2, 4, 0, 2, 1);
cout << "Test 7.6(0ocurrancekey):  " << (test1MergeKeyOut7.AssertEquals("[]") == 1 ? "true" : "false") << endl;
test1MergeKeyOut7.prinf();

Sequence<int, int> test1MergeKeyOut8 = merge_key(test1MergePos, 3, 20, 1, test1MergePos2, 4, 20, 2, 1);
cout << "Test 7.7(bigocurrancekey):  " << (test1MergeKeyOut8.AssertEquals("[]") == 1 ? "true" : "false") << endl;
test1MergeKeyOut8.prinf();

Sequence<int, int> test1MergeKeyOut9 = merge_key(test1MergePos, 55, 2, 1, test1MergePos2, 55, 1, 2, 1);
cout << "Test 7.8(noexistkey):  " << (test1MergeKeyOut9.AssertEquals("[]") == 1 ? "true" : "false") << endl;
test1MergeKeyOut9.prinf();
```

```
Sequence<int, int> test1MergeKeytest;
Sequence<int, int> test1MergeKeytest2;

Sequence<int, int> test1MergeKeyOut10 = merge_key(test1Mergetest, 3, 2, 1, test1Mergetest2, 4, 2, 2, 1);
cout << "Test 7.9:(nullsequences) " << (test1MergeKeyOut10.AssertEquals("[]") == 1 ? "true" : "false") << endl;
test1MergeKeyOut10.prinf();

Sequence<int, int> test1MergeKeyOut11 = merge_key(test1MergePos, 2, -2, -1, test1MergePos2, 2, -1, -2, -1);
cout << "Test 7.10(negativevalues):  " << (test1MergeKeyOut11.AssertEquals("[]") == 1 ? "true" : "false") << endl;
test1MergeKeyOut11.prinf();

Sequence<int, int> test1MergeKeyOut12 = merge_key(test1MergePos, 4, 1, 1, test1MergePos2, 2, -1, -2, 3);
cout << "Test 7.11(onenegative):  " << (test1MergeKeyOut12.AssertEquals("[4,4][5,5][6,6]") == 1 ? "true" : "false") << endl;
test1MergeKeyOut12.prinf();
```

Output:

```
--------MERGE BY KEY---------
Test 7.1: true
[3,50][4,40][5,50][6,60][7,70]
Test 7.2(biggerLength):  true
[3,50][4,40][5,50][6,60][7,70][8,80][9,90]
Test 7.3(0length):  true
[]
Test 7.4(0count):  true
[]
Test 7.5(1count):  true
[3,50][4,40][5,50]
Test 7.6(0ocurrancekey):  true
[]
Test 7.7(bigocurrancekey):  true
[]
Test 7.8(noexistkey):  true
[]
Test 7.9:(nullsequences) true
[]
Test 7.10(negativevalues):  true
[]
Test 7.11(onenegative):  true
[4,4][5,5][6,6]
```

Merge by key tests:

Code:

```
Sequence<int, int> testemptyandkey;

cout << "Test 8.1:  " << (testemptyandkey.isEmpty()== true ? "true" : "false") << endl;
testemptyandkey.insertAtEnd(0, 0);
cout << "Test 8.2:  " << (testemptyandkey.itsNodeEmpty(1)== false ? "true" : "false") << endl;
testemptyandkey.insertAtEnd(1, 1);
testemptyandkey.insertAtEnd(2, 2);
testemptyandkey.insertAtEnd(3, 3);
testemptyandkey.insertAtEnd(4, 4);
testemptyandkey.insertAtEnd(5, 5);

cout << "Test 8.3:  " << (testemptyandkey.isEmpty()== false ? "true" : "false") << endl;
cout << "Test 8.4:  " << (testemptyandkey.itsNodeEmpty(5)== true ? "true" : "false") << endl;
cout << "Test 8.5:  " << (testemptyandkey.itsNodeEmpty(6)== false ? "true" : "false") << endl;
cout << "Test 8.6:  " << (testemptyandkey.getKey(3) == 3 ? "true" : "false") << endl;
cout << "Test 8.7:  " << (testemptyandkey.getKey(5) == 5 ? "true" : "false") << endl;
cout << "Test 8.8:  " << (testemptyandkey.getKey(0) == 0 ? "true" : "false") << endl;
```

Output:

```
--------IsEmpty, getKey and hasnext ---------
Test 8.1:  true
Test 8.2:  true
Test 8.3:  true
Test 8.4:  true
Test 8.5:  true
Test 8.6:  true
Test 8.7:  true
Test 8.8:  true
```