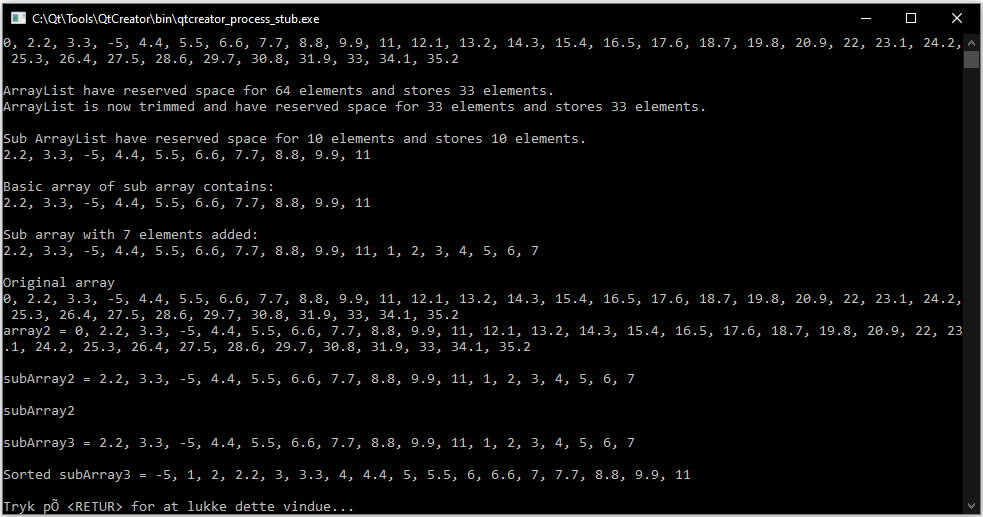
Lektion 12: Opgave 18.1 (TA)

# Output:



# Kildekode:

## ArrayList.h

#ifndef ARRAYLIST\_H

#define ARRAYLIST\_H

#include <iostream>

template <typename T>

class ArrayList {

public:

*// Default constructor, initialized as null*

    ArrayList() {

        mSize = 0;

        mReserved = 0;

        mElems = nullptr;

    }

*// Copy constructor*

    ArrayList(const ArrayList<T>& *c*) {

        mSize = c.mSize;*// Set attributes from Arraylist c to be equal to the new objects attributes*

        mReserved = c.mReserved;

        if (mSize > 0) {

            mElems = new T[mReserved];*// Reserve space for the new object*

        }

        for (int i = 0; i < mSize; ++i) {

            mElems[i] = c.mElems[i];*// Set the elements in the new object equal to those of Arraylist c*

        }

    }

*// Move constructor*

    ArrayList(ArrayList<T>&& *c*) {

        mElems = c.mElems;*// Set attributes from Arraylist c to be equal to the new objects attributes*

        mSize = c.mSize;

        mReserved = c.mReserved;

        c.mElems = nullptr;*// Remove old Arraylist (Arraylist c)*

        c.mSize = 0;

        c.mReserved = 0;

    }

*// Constructor with initialization of "initialized" elements*

    ArrayList(int *initialized*) {

        if (initialized <= 0) {

            throw std::invalid\_argument("Initialization has to be larger than zero!");

        }

        mSize = 0;

        mReserved = initialized;*// Set the reserved member variable to the initialized size*

        mElems = new T[mReserved];*// Reserve space for the new object with a given initialized size*

    }

*// Destructor*

    virtual ~ArrayList() {

        if (mReserved > 0) {

            delete[] mElems;*// Destructs the object by deleting the reserved space*

        }

    }

*// Copy assignment operator*

    ArrayList<T >& operator=(const ArrayList<T>& *a*) {

        if (mReserved > 0) {

            delete[] mElems;*// Deletes contents of member variable mElems*

        }

        mSize = a.mSize;*// Sets attributes to be equal to ArrayList a's attributes*

        mReserved = a.mReserved;

        if (mSize > 0) {

            mElems = new T[mReserved];*// Reserve new space equal to the size of ArrayList a*

        }

        for (int i = 0; i < mSize; ++i) {

            mElems[i] = a.mElems[i];*// Copy the elements of ArrayList a to the new object*

        }

        return \*this;

    }

*// Move assignment operator*

    ArrayList <T>& operator=(ArrayList <T>&& *a*) {

        if (mReserved > 0) {

            delete[] mElems;*// Deletes contents of member variable mElems*

        }

        mElems = a.mElems;*// Set attributes from Arraylist a to be equal to the new objects attributes*

        mSize = a.mSize;

        mReserved = a.mReserved;

        a.mElems = nullptr;*// Remove old Arraylist (Arraylist a)*

        a.mSize = 0;

        a.mReserved = 0;

        return \*this;

    }

*// Add element to dynamic array*

    void add(const T& *element*) {

        if (mSize == mReserved)*// If the ArrayList is full extend storage*

            extendStorage();

        mElems[mSize] = element;*// Add the new element to the end of the ArrayList*

        ++mSize;*// Update the size of the ArrayList*

    }

*/\**

*\* Inserts the element at placement "idx" in array and moves the remaining*

*\* items by one place, restoring the old element at "idx".*

*\* check whether it is needed to extend the storage .*

*\* move all elements from mSize to idx (reverse) one element to the right in the array*

*\* set mElems [idx] equal to the element to be inserted*

*\*/*

    void add(int *idx*, const T& *element*) {

        if (idx <= 0) {

            throw std::invalid\_argument("Index has to be larger than zero!");

        }

        if (mSize == mReserved)*// If the ArrayList is full extend storage*

            extendStorage();

        for (int i = mSize + 1; i > idx; --i) {

            mElems[i] = mElems[i - 1];*// Moves all elements one element to the right*

        }

        mElems[idx] = element;*// Add the new element to the ArrayList at idx*

        ++mSize;*// Update the size of the ArrayList*

    }

*// Get a const reference to the element at idx*

    const T& operator[](int *idx*) const {

        if (idx > mSize || idx < 0) {

            throw std::invalid\_argument("Index out of range!");

        }

        return mElems[idx];*// Retruns a const reference*

    }

*// Get a reference to the element at idx*

    T& operator[](int *idx*) {

        if (idx > mSize || idx < 0) {

            throw std::invalid\_argument("Index out of range!");

        }

        return mElems[idx];*// Retruns a reference*

    }

*/\**

*\* Removes the element at placement "idx" by moving all the remaining elements*

*\* by one place to the left in the array*

*\*/*

    void remove(int *idx*) {

        if (idx > mSize || idx < 0) {

            throw std::invalid\_argument("Index out of range!");

        }

        for (int i = idx; i < mSize; i++) {

            mElems[i] = mElems[i + 1];*// Moves remaining elements to the left overwriting idx*

        }

        --mSize;*// Update the size of the ArrayList*

    }

*// Returns the number of elements stored*

    int size() const { return mSize; }

*// Returns the number of items currently reserved inmemory*

    int reserved() const { return mReserved; }

*// Returns true if number of elements in array is zero*

    bool isEmpty() const { return (mSize == 0) ? true : false; }

*// Trims the storage array to the exact number of elements stored.*

    void trimToSize() {

        mReserved = mSize;*// Updates the reserved member variable to be equale to the actual size*

        T\* tmp = new T[mReserved];*// Reserves space in temporary variable*

        for (int i = 0; i < mSize; i++) {

            tmp[i] = mElems[i];*// Copies elements into tmp*

        }

        delete[] mElems;*// Delets elements in original ArrayList*

        mElems = new T[mReserved];*// Reserves space equal to the actual size of the array*

        for (int i = 0; i < mSize; i++) {

            mElems[i] = tmp[i];*// Copies elements back into the newly reserved space*

        }

        delete[] tmp;*// Deletes the temporary variable*

    }

*/\**

*\* Sorts the array using insertion sort (or another algorithm)*

*\* You are not allowed to use standard algorithms from algorithm header.*

*\*/*

    void sort() {

        T key;

        int i, j;

            for (i = 1; i < mSize; i++) {*// Move elements of arr[0..i-1], that are greater than key,*

                key = mElems[i];*// to one position ahead of their current position*

                j = i - 1;

                while (j >= 0 && mElems[j] > key) {

                    mElems[j + 1] = mElems[j];

                    j = j - 1;

                }

                mElems[j + 1] = key;

            }

    }

*// Returns a new ArrayList with elements from "fromIdx" index to "toIdx"*

    ArrayList<T> subArrayList(int *fromIdx*, int *toIdx*) const {

        if (fromIdx > toIdx) {

            throw std::invalid\_argument("fromIdx is larger than toIdx");

        } else if (fromIdx == toIdx) {

            throw std::invalid\_argument("fromIdx and toIdx is equal");

        } else if (fromIdx < 0 || toIdx < 0) {

            throw std::invalid\_argument("fromIdx or toIdx is less than zero");

        } else if (fromIdx > mSize || toIdx > mSize) {

            throw std::invalid\_argument("fromIdx or toIdx is larger than the size of the ArrayList");

        }

        ArrayList<T> array((toIdx - fromIdx) + 1);*// Create a new ArrayList with the reserved size of the differance between the indexes*

        for (int i = fromIdx, j = 0; i <= toIdx; ++i, ++j) {

            array.mElems[j] = mElems[i];*// Copy the elements to the new ArrayList*

        }

        array.mSize = array.mReserved;*// Set the new ArrayList's size attribute*

        return array;*// Return the new ArrayList*

    }

*// Returns a new C style array (copy created with new) with all elements*

    T\* toArray() {

        T\* cArray = new T[mSize];*// Reserves space for C style array*

        for (int i = 0; i < mSize; i++) {

            cArray[i] = mElems[i];*// Copies the elements into the newly reserved space*

        }

        return cArray;*// Returns the C style array*

    }

private:

*/\**

*\* extendStorage():*

*\* create new array with size 2\* mReserved*

*\* copy old data to the new array*

*\* delete old array*

*\* update pointer mElems to point to the new array*

*\* (Since this method is private, the method will only be used internally,*

*\* but the functionality is needed).*

*\*/*

    void extendStorage() {

        mReserved = (mReserved == 0) ? 1 : mReserved \* 2;*// If the reserved space is 0, set it to 1, else multiply it with 2*

        T\* tmp = new T[mReserved];*// Reserves space in temporary variable*

        for (int i = 0; i < mSize; i++) {

            tmp[i] = mElems[i];*// Copies elements into tmp*

        }

        delete[] mElems;*// Delets elements in original ArrayList*

        mElems = new T[mReserved];*// Reserves space equal to the new size of the array*

        for (int i = 0; i < mSize; i++) {

            mElems[i] = tmp[i];*// Copies elements back into the newly reserved space*

        }

        delete[] tmp;*// Deletes the temporary variable*

    }

*// Member variables*

    int mReserved;*// The current capacity of " mElems " array*

    int mSize;*// The number of elements stored*

    T\* mElems;*// Array for storing the elements*

};

# endif*// ARRAYLIST\_H*

## main.cpp

#include <iostream>

#include "ArrayList.h"

int main () {

    ArrayList<double> array;

    for (int i = 0; i < 33; ++i) {

        array.add (i \* 1.1);

    }

    array.add(4, -5);

    array.remove(1);

    for (int i = 0; i < array.size(); ++i) {

        std::cout << array[i] << ", ";

    }

    std::cout << "\b\b " << std::endl << std::endl;

    std::cout << "ArrayList have reserved space for " << array.reserved()

              << " elements and stores " << array.size () << " elements." << std::endl;

    array.trimToSize();

    std::cout << "ArrayList is now trimmed and have reserved space for " << array.reserved()

              << " elements and stores " << array.size() << " elements." << std::endl << std::endl;

    ArrayList<double> subArray = array.subArrayList(1 ,10);

    std::cout << "Sub ArrayList have reserved space for " << subArray.reserved()

              << " elements and stores " << subArray.size() << " elements." << std::endl;

    for (int i = 0; i < subArray.size(); ++i) {

        std::cout << subArray[i] << ", ";

    }

    std::cout << "\b\b " << std::endl << std::endl;

    double\* sArray = subArray.toArray();

    for (int i = 0; i < 7; ++i) {

        subArray.add(i+1);

    }

*// The basic array prints after adding elements to subArray*

    std::cout << "Basic array of sub array contains: " << std::endl;

    for (int i = 0; i < subArray.size() - 7; ++i) {

        std::cout << sArray[i] << ", ";

    }

    std::cout << "\b\b " << std::endl << std::endl;

*// The sub array now has elements*

    std::cout << "Sub array with 7 elements added: " << std::endl;

    for (int i = 0; i < subArray.size(); ++i) {

        std::cout << subArray[i] << ", ";

    }

    std::cout << "\b\b " << std::endl << std::endl;

*// The original array has*

    std::cout << "Original array " << std::endl;

    for (int i = 0; i < array.size(); ++i) {

        std::cout << array[i] << ", ";

    }

    std::cout << "\b\b " << std::endl;

*// Copy constructor*

*// Copy assignment*

    ArrayList<double> array2 = array;

    ArrayList<double> subArray2;

    subArray2 = subArray;

    std::cout << "array2 = ";

    for (int i = 0; i < array2.size(); ++i) {

        std::cout << array2[i] << ", ";

    }

    std::cout << "\b\b " << std::endl << std::endl;

    std::cout << "subArray2 = ";

    for (int i = 0; i < subArray2.size(); ++i) {

        std::cout << subArray2[i] << ", ";

    }

    std::cout << "\b\b " << std::endl << std::endl;

*// Move assignment operator*

    ArrayList<double> subArray3 = std::move(subArray2);

    array2 = std::move(subArray2);

    std::cout << "subArray2 = ";

    for (int i = 0; i < subArray2.size(); ++i) {

        std::cout << subArray2[i] << ", ";

    }

    std::cout << "\b\b " << std::endl << std::endl;

    std::cout << "subArray3 = ";

    for (int i = 0; i < subArray3.size(); ++i) {

        std::cout << subArray3[i] << ", ";

    }

    std::cout << "\b\b " << std::endl << std::endl;

*// Insertion sort*

    subArray3.sort();

    std::cout << "Sorted subArray3 = ";

    for (int i = 0; i < subArray3.size(); ++i) {

        std::cout << subArray3[i] << ", ";

    }

    std::cout << "\b\b " << std::endl << std::endl;

    return 0;

}