Printout for cs320-09-T-A03-VectorT.hpp

```
// File: VectorT/VectorT.hpp

#ifndef VECTORT_HPP_
#define VECTORT_HPP_

#include <iostream> // istream, ostream.
#include "ASeq.hpp"
using namespace std;

// ========= VectorT =========
// Pre: The parameter T is not a pointer type.
template<class T>
class VectorT : public ASeq<T> {
private:
    T *_data;
    int _cap; // Invariant: 0 < _cap, and _cap is a power of 2.
    int _size; // Invariant: 0 <= _size <= _cap.

    void doubleCapacity();

    VectorT(VectorT const &rhs); // Disabled.
    VectorT &operator=(VectorT const &rhs); // Disabled.

public:
    VectorT();
    // Post: This vector is initialized with capacity of 1 and size of 0.

    virtual ~VectorT();

    void append(T const &e);
    // Post: Element e is appended to this vector, possibly increasing cap().

    int cap() const  override {return _cap;}
    // Post: The capacity of this vector is returned.

    void fromStream(istream &is);
    // Post: The items of input stream is are appended to this vector.

    void insert(int i, T const &e);
    // Pre: 0 <= i && i <= size().
    // Post: Items [i..size()-1] are shifted right and element e is inserted at position
i.
    // size() is increased by 1, possibly increasing cap().

    T &operator[](int i) override; // For read/write.
    T const &operator[](int i) const override; // For read-only.

    T remove(int i);
    // Pre: 0 <= i && i < size(). T has a copy constructor.
    // Post: Element e is removed from position i and returned.
    // Items [i+1..size()-1] are shifted left.
    // size() is decreased by 1 (and cap() is unchanged).

    int size() const {return _size;}
    // Post: The size of this vector is returned.

    void toStream(ostream &os) const;
    // Post: A string representation of this vector is returned to output stream os.
};

// ========= Constructor =========
template<class T>
VectorT<T>::VectorT() {
    _data = new T[1];
```

```
    _cap = 1;
    _size = 0;
}

// ========= Destructor =========
template<class T>
VectorT<T>::~VectorT() {
    delete [] _data; // Differs for VectorP.
    _data = nullptr;
}

// ========= append =========
template<class T>
void VectorT<T>::append(T const &e) {
    if (_size == _cap) {
        doubleCapacity();
    }
    _data[_size++] = e;
}

// ========= doubleCapacity =========
template<class T>
void VectorT<T>::doubleCapacity() {
    _cap *= 2;
    T *newDat = new T[_cap];
    for (int k = 0; k < _size; k++) {
        newDat[k] = _data[k];
    }
    delete[] _data;
    _data = newDat;
}

// ========= fromStream =========
template<class T>
void VectorT<T>::fromStream(istream &is) {
    T temp;
    while (is >> temp) {
        append(temp);
    }
}

// ========= insert =========
template<class T>
void VectorT<T>::insert(int i, T const &e) {
    int k = i;
    if(_cap == _size){
        doubleCapacity();
    }
    for (int j = _size; j != k; j--) {
        _data[j] = _data[j-1];
    }
    _size++;
    _data[i] = e;
}

// ========= operator[] =========
template<class T>
T &VectorT<T>::operator[](int i) {
    if (i < 0 || _size <= i) {
        cerr << "VectorT index out of bounds: index == " << i << endl;
        throw -1;
    }
    return _data[i];
}

template<class T>
```

```
T const &VectorT<T>::operator[](int i) const {
    if (i < 0 || _size <= i) {
        cerr << "VectorT index out of bounds: index == " << i << endl;
        throw -1;
    }
    return _data[i];
}

// ========= operator<< =========
template<class T>
ostream &operator<<(ostream &os, VectorT<T> const &rhs) {
    rhs.toStream(os);
    return os;
}

// ========= operator>> =========
template<class T>
istream &operator>>(istream &is, VectorT<T> &rhs) {
    rhs.fromStream(is);
    return is;
}

// ========= remove =========
template<class T>
T VectorT<T>::remove(int i) {
    int temp = _data[i];
    for (int j = i; j < (_size-1); j++){
        _data[j] = _data[j+1];
    }
    _size--;
    return temp;
}

// ========= toStream =========
template<class T>
void VectorT<T>::toStream(ostream &os) const {
    os << "(";
    for (int i = 0; i < _size - 1; i++) {
        os << _data[i] << ", ";
    }
    if (_size > 0) {
        os << _data[_size-1];
    }
    os << ")";
}

#endif

// new page
```