#include <iostream>

using namespace std;

// Template class untuk array dinamis

template <class T>

class Array1Ddinamis {

public:

    Array1Ddinamis(int sz);

    ~Array1Ddinamis();

    Array1Ddinamis<T>& ReSize(int sz);

    T& operator[](int index);

    const T& operator[](int index) const;

    int Size() const;

    // Friend function untuk operator overloading

    template <class U>

    friend ostream& operator<<(ostream& os, const Array1Ddinamis<U>& arr);

    template <class U>

    friend istream& operator>>(istream& is, Array1Ddinamis<U>& arr);

private:

    T\* elements\_;

    int size\_;

};

template <class T>

Array1Ddinamis<T>::Array1Ddinamis(int sz) : size\_(sz) {

    elements\_ = new T[sz];

}

template <class T>

Array1Ddinamis<T>::~Array1Ddinamis() {

    delete[] elements\_;

}

template <class T>

Array1Ddinamis<T>& Array1Ddinamis<T>::ReSize(int sz) {

    if (sz == size\_) return \*this;

    T\* newData = new T[sz];

    int minSize = (sz < size\_) ? sz : size\_;

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

        newData[i] = elements\_[i];

    }

    delete[] elements\_;

    elements\_ = newData;

    size\_ = sz;

    return \*this;

}

template <class T>

T& Array1Ddinamis<T>::operator[](int index) {

    if (index >= 0 && index < size\_) {

        return elements\_[index];

    } else {

        throw out\_of\_range("Index out of range");

    }

}

template <class T>

const T& Array1Ddinamis<T>::operator[](int index) const {

    if (index >= 0 && index < size\_) {

        return elements\_[index];

    } else {

        throw out\_of\_range("Index out of range");

    }

}

template <class T>

int Array1Ddinamis<T>::Size() const {

    return size\_;

}

// Operator overloading untuk output

template <class T>

ostream& operator<<(ostream& os, const Array1Ddinamis<T>& arr) {

    for (int i = 0; i < arr.Size(); ++i) {

        os << "Element [" << i << "] = " << arr.elements\_[i] << endl;

    }

    return os;

}

// Operator overloading untuk input

template <class T>

istream& operator>>(istream& is, Array1Ddinamis<T>& arr) {

    for (int i = 0; i < arr.Size(); ++i) {

        cout << "Masukkan nilai untuk elemen [" << i << "]: ";

        is >> arr.elements\_[i];

    }

    return is;

}

int main() {

    int initialSize;

    // Input ukuran awal array

    cout << "Masukkan ukuran awal array: ";

    cin >> initialSize;

    Array1Ddinamis<int> myArray(initialSize);

    // Mengisi array dengan nilai

    cout << "Masukkan nilai untuk array: " << endl;

    cin >> myArray;

    // Menampilkan nilai array

    cout << "Array Dinamis Sebelum Resize: " << endl;

    cout << myArray;

    // Mengubah ukuran array

    int newSize;

    cout << "Masukkan ukuran baru array: ";

    cin >> newSize;

    myArray.ReSize(newSize);

    // Jika ukuran baru lebih besar, meminta input untuk elemen tambahan

    if (newSize > initialSize) {

        cout << "Masukkan nilai tambahan untuk array: " << endl;

        cin >> myArray;

    }

    // Menampilkan nilai array setelah resize

    cout << "Array Dinamis Setelah Resize: " << endl;

    cout << myArray;

    return 0;

}