#include<iostream>

#include<initializer\_list>

using namespace std;

//template<typename T,int size>

//class FixedArray {

// T arr[size]{};

// size\_t mysize = 0;

//public:

// FixedArray(initializer\_list<T> list) {

//

// for (auto data : list) {

// arr[mysize] = data;

// ++mysize;

// }

//

//

// }

//

// void Show()const {

// for (size\_t i = 0; i < mysize; i++)

// {

// cout << arr[i] << " ";

// } cout << endl;

// }

//

// T& operator[](int index) {

// return arr[index];

// }

//

//};

//

//

//void main() {

// FixedArray<int, 20>myarray{ 1,2,3,4,1,1,1,1,1,1,1,1,1 };

// myarray.Show();

//

//}

//Dynamic Data Structure

//Stack

//Queue

//Dequeue

//Circular

//Binary Tree

//Linked List

//Double Linked List

#include<assert.h>

class Stack {

int\* data;

size\_t size = 0;

public:

Stack():data(NULL),size(NULL){}

void push(int value) {

cout << value << " value added successfully" << endl;

auto newarray = new int[size+1] {};

for (size\_t i = 0; i < size; i++)

{

newarray[i] = data[i];

}

newarray[size] = value;

if (size != 0) {

delete[]data;

}

data = newarray;

newarray = nullptr;

++size;

}

int pop() {

assert(size > 0);

auto newarray = new int[size-1] {};

for (size\_t i = 0; i < size-1; i++)

{

newarray[i] = data[i];

}

int last = data[size - 1];

if (size != 0) {

delete[]data;

}

data = newarray;

newarray = nullptr;

--size;

return last;

}

int peek() {

assert(size > 0);

return data[size - 1];

}

int GetSize()const {

return size;

}

void Clear() {

if (this->data == NULL || this->size == NULL) {

return;

}

delete[]data;

this->data = NULL;

this->size = NULL;

}

~Stack()

{

delete[]data;

}

};

void main() {

Stack mystack;

mystack.push(10);

mystack.push(11);

mystack.push(12);

cout << mystack.peek() << endl;

mystack.pop();

cout << mystack.peek() << endl;

//cout << mystack.pop() << endl;

//cout << mystack.pop() << endl;

//

}