#include<iostream>

#include<initializer\_list>

using namespace std;

//Queue

//Dequeue

//Circular

//Binary Tree

//Linked List

//Double Linked List

//class Queue {

// int\* arr;

// int capacity;

// int front;

// int rear;

// int count;

//public:

// Queue(int size) {

// arr = new int[size] {};

// capacity = size;

// front = 0;

// rear = -1;//you think like index

// count = 0;

// }

//

// //enqueue dequeue peek isEmpty isFull

//

//

// void enqueue(int value) {

// assert(!IsFull() && "Queue is full");

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

// arr[++rear] = value;

// ++count;

// }

//

// void dequeue() {

// assert(!IsEmpty() && "Queue is empty");

//

// auto temp = new int[count - 1]{};

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

// {

// temp[i] = arr[i + 1];

// }

// if (!IsEmpty()) {

// delete[]arr;

// }

// arr = temp;

// temp = nullptr;

// --count;

// }

//

// int peek() {

// return arr[front];

// }

//

// bool IsEmpty()const {

// return size() == 0;

// }

//

// bool IsFull()const {

// return size() == capacity;

// }

// int size()const {

// return count;

// }

//

// ~Queue()

// {

// delete[]arr;

// }

//

//};

//

//template<typename T>

//class Queue {

// T\* arr;

// int capacity;

// int front;

// int rear;

// int count;

//public:

// Queue(int size) {

// arr = new T[size]{};

// capacity = size;

// front = 0;

// rear = -1;//you think like index

// count = 0;

// }

//

// //enqueue dequeue peek isEmpty isFull

//

//

// void enqueue(T value) {

// assert(!IsFull() && "Queue is full");

// arr[++rear] = value;

// ++count;

// }

//

// void dequeue() {

// assert(!IsEmpty() && "Queue is empty");

//

// auto temp = new T[count - 1]{};

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

// {

// temp[i] = arr[i + 1];

// }

// if (!IsEmpty()) {

// delete[]arr;

// }

// arr = temp;

// temp = nullptr;

// --count;

// }

//

// T peek() {

// return arr[front];

// }

//

// bool IsEmpty()const {

// return size() == 0;

// }

//

// bool IsFull()const {

// return size() == capacity;

// }

// int size()const {

// return count;

// }

//

// ~Queue()

// {

// delete[]arr;

// }

//

//};

#include<assert.h>

template<typename T>

class Stack {

T\* data;

size\_t size = 0;

public:

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

void push(T value) {

auto newarray = new T[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;

}

T pop() {

assert(size > 0);

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

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

{

newarray[i] = data[i];

}

T last = data[size - 1];

if (size != 0) {

delete[]data;

}

data = newarray;

newarray = nullptr;

--size;

return last;

}

T 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;

}

};

class Person {

string name;

string surname;

double money;

public:

Person() = default;

Person(const string& n, const string& s, const double& m) {

SetName(n);

SetSurname(s);

SetMoney(m);

}

string GetName()const {

return name;

}

string GetSurname()const {

return surname;

}

double GetMoney()const {

return money;

}

void SetName(const string& n) {

this->name = n;

}

void SetSurname(const string& s) {

this->surname = s;

}

void SetMoney(const double& m) {

if (m >= 0)

this->money = m;

}

friend ostream& operator<<(ostream& out, const Person& p) {

out << "Person INFO : " << endl;

out << "Name : " << p.GetName() << endl;

out << "Surname : " << p.GetSurname() << endl;

out << "Money : " << p.GetMoney() << endl;

return out;

}

};

class Worker {

};

class Bank {

Stack<Person> persons;

Stack<Worker> workers;

public:

void AddPerson(const Person& p) {

persons.push(p);

}

void AddWorker(const Worker& w) {

workers.push(w);

}

void ShowAll() {

int size = persons.GetSize();

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

{

auto person = persons.peek();

cout << person;

persons.pop();

}

}

};

void main() {

Person p1("Elvin", "Camalzade", 0);

Person p2("Emil", "Eliyev", 20);

Person p3("Azad", "Ahmedzade", 30);

Bank bank;

bank.AddPerson(p1);

bank.AddPerson(p2);

bank.AddPerson(p3);

bank.ShowAll();

}