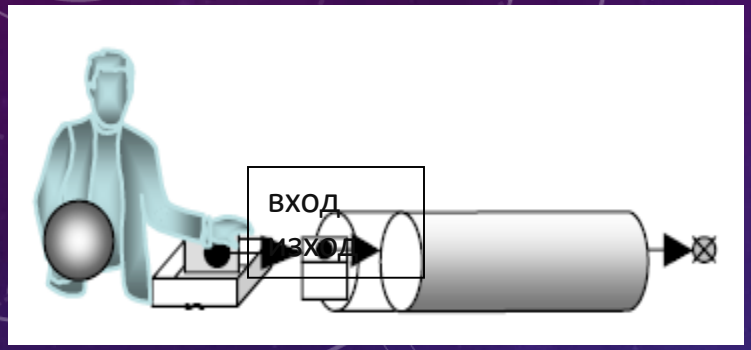
Национална школа, Враца-2025г.

АБСТРАКТНИ СТРУКТУРИ ОТ ДАННИ. СТЕК И ОПАШКА



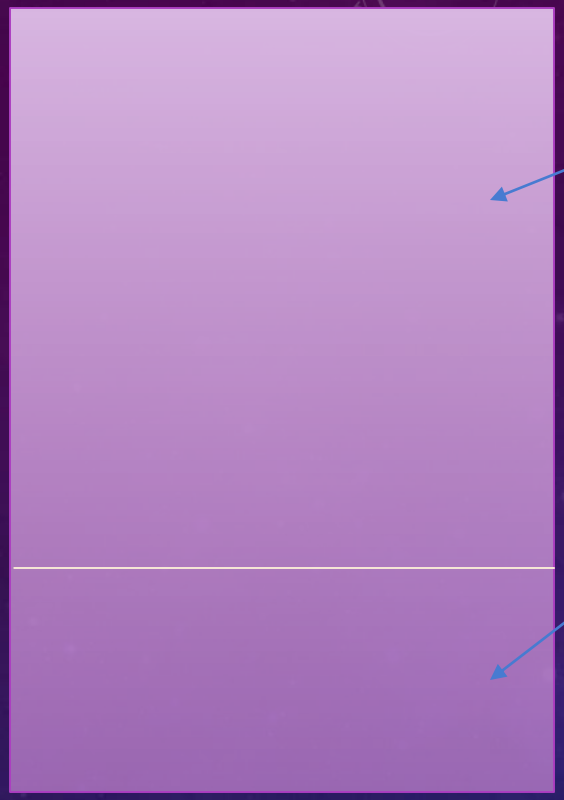
Лектор:
Бистра Танева
МГ „Акад. Кирил Попов“

1. ОПРЕДЕЛЕНИЕ – Начин за организиране и съхраняване на данните в паметта.

2. Различни структури от данни според достъпа до елементите:

- с пряк достъп – масив, вектор
- с ограничен достъп – стек, опашка, дек, списък и др.

3. Тип Vector – тип данни, който резлизира динамичен масив , за разлика от масива, където броя на елементите е фиксиран, тук той е променлив- може да добавяме и махаме елементи.



Динамична памет -заделя се по време на изпълнение на програмата според необходимостта.

Статична памет – всички променливи, които дефинира програмата.

Първата стъпка към използването на вектор е да добавим неговата библиотека:
#include<vector>
Разбира се, ако вече сме добавили библиотека, която го включва, това не е необходимо. Например може да използваме:
#include<bits/stdc++.h>

А) Деклариране на вектор - vector<тип> име;

vector <int> v;

В статичната памет се заделят 4 байта място, но ако не стигне автоматично се заделя място двойно повече в динамичната памет.

Броят на елементите, които в момента са във вектор, може да се достъпи чрез функцията size:

v.size(); // броя на елементите във вектора v

Б) Достъп до елементите

Достъпът до елементи във вектор става по същия начин както в масив. Всеки елемент има позиция от 0 до v.size()-1.

Така самите елементи са: v[0], v[1], ..., v[v.size()-1].

В) Въвеждане на елементи

```
cin>>x;
v.push_back(x); //добавяме го в края на вектора.
v.pop_back(); // премахва последния елемент.
```

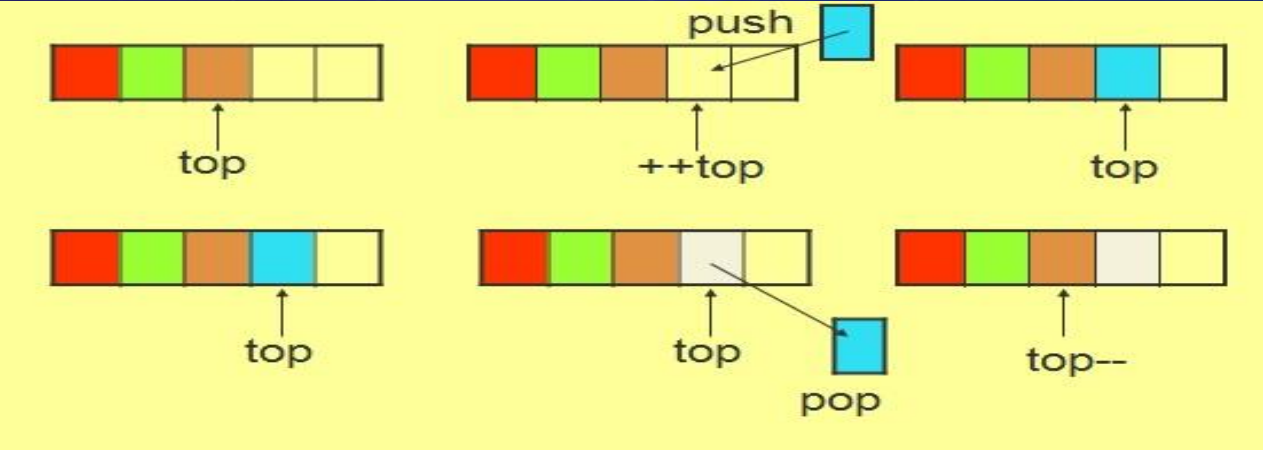
Пример:

Декларираме вектор. Добавяме елементи във вектор до въвеждане на числото 0. Извеждане на елементите на един ред.

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{
    vector<int> v;
    int x,n;
    do
    {
        cin>>x;
        if(x!=0)v.push_back(x);
    }
    while(x!=0);
    n=v.size();
    for(int i=0;i<n;i++)
    {
        cout<<v[i]<<' ';
    }
    cout<<endl;
    return 0;
}
```

4. СТЕК

- Стекът е абстрактен тип данни, който съхранява елементи от произволен тип.
- Включването и изключването на елемент се извършва само в единия му край по схемата Last-In-First-Out (LIFO).
- Прието е елементът, върху който се извършват операциите, да се нарича връх на стека



ОПЕРАЦИИ СЪС СТЕК

- Основни операции - добавяне на елемент на върха на стека и извличане на елемент от върха на стека.

- Допълнителни операции – проверка за празен стек, достъп до елемент на върха, извличане на елемент без изключването му, добавяне/изключване на *к* елемента, връщане броя на елементи в стека

ФУНКЦИИ ОТ СТАНДАРТНАТА БИБЛИОТЕКА

`#include <stack>`	
`stack <typename> s`	s - стек от елементи от тип typename
`s.push(x)`	включване на елемент x в стека
`s.pop()`	премахване на елемента от върха на стека
`s.top()`	достъп до елемента във върха на стека
`s.empty()`	проверка дали стекът е празен
`s.size()`	връща броя на елементите в стека

УПРАЖНЕНИЕ

Задача 1.

В празен стек последователно постъпват елементите 2, 5, 7, 12, 1, 10:

а) Начертайте схема, отразяваща разположението на елементите в стека.

б) До кой елемент от стека има достъп?

в) До кой елемент от стека има достъп след отстраняването на два елемента?

г) До кой елемент от стека има достъп след отстраняването на един елемент, постъпването на елементите 4, 8 и отстраняването на четири елемента.

Направете програма, която да описва действията в примера.

```
#include <iostream>
#include <stack>
using namespace std;
{
stack <int> s;
s.push(2);
s.push(5);
s.push(7);
s.push(12);
s.push(1);
s.push(10);
cout<<s.top()<<endl;
s.pop();
s.pop();
cout<<s.top()<<endl;
s.pop();
s.push(4);
s.push(8);
s.pop();
s.pop();
s.pop();
s.pop();
cout<<s.top()<<endl;
return 0;
}
```

ЗАДАЧА 2.

Въведи дума от клавиатурата. Изведи я обратно (в обратен ред), използвайки стек.

Пример:

Вход: kotka

Изход: aktok

☞ *Обяснение:* Всеки символ се поставя в стека, а после се извежда обратно – така се обръща редът.

```
#include <iostream>
#include <stack>
using namespace std;
int main() {
    string duma;
    stack<char> s;
 cout << "Vavedi duma: ";
    cin >> duma;
// Слагаме буквите в
стека
for (int i = 0; i <
duma.size(); i++)
    {
char bukva = duma[i];
 s.push(bukva); }

cout << "Obarnata duma: ";
    while (!s.empty())
 {
        cout << s.top();
// показваме последната
буква
        s.pop();
// махаме я от стека
    }
 cout << endl;
  return 0;
}
```

ЗАДАЧА 3.

Да се провери дали дума е палиндром (чете се по един и същи начин отпред и отзад), като се използва стек.

Пример:
Вход: око → YES
Вход: маса → NO

☞ *Обяснение:*

- Въвеждаме дума от потребителя.
- Създаваме стек за символи.
- Добавяме всеки символ от думата в стека.
- Достъпваме и махаме последния символ от стека.
- Ако символът от началото не съвпада със символа от края → не е палиндром.

```
int main() {
    string duma;
    stack<char> s;

 cout << "Vavedi duma:;
    cin >> duma;

    // Слагаме буквите в
стека
  for (char bukva : duma)
{
  s.push(bukva);
    }
```

```
 // Сравняваме с буквите
отзад
    bool palindrom = true;
  for (char bukva : duma)
{
    if (bukva != s.top())
{
     palindrom = false;
       break;
 }
        s.pop();
 }
  if (palindrom)
   cout << YES" << endl;
  else
  cout << “ NO" << endl;
    }
```

ЗАДАЧА 4. Проверка на правилно записани скоби

Дадена е последователност от скоби — отварящи и затварящи: (и).

Напишете програма, която проверява дали скобите са правилно записани.

Какво значи "правилно записани скоби"?

- Всяка отваряща скоба (има своя затваряща скоба).
- Скобите са добре вложени — например:
  - ()() е правилно
  - (()) е правилно
  - (() не е правилно (липсва затваряща скоба)
  - ())(не е правилно

Какво прави стекът в примера){{[]]

1.(→ push

2.) → pop → съвпада

3.{ → push

4.{ → push

5.[→ push

6.] → pop → съвпада

7.] → pop → ✕ не съвпада (взима {, а трябва да е]) → **връща false**

```
#include <iostream>
#include <stack>
#include <string>
using namespace std;

// Функция, която проверява дали скобите са правилни
bool pravilniSkobi(string tekst) {
    stack<char> s; // стек за отварящи скоби

    for (int i = 0; i < tekst.length(); i++) {
        char znak = tekst[i];

        // Ако е отваряща скоба - запомняме я
        if (znak == '(' || znak == '[' || znak == '{') {
            s.push(znak);
        }
        // Ако е затваряща скоба - проверяваме дали съвпада
        else if (znak == ')' || znak == ']' || znak == '}') {
            if (s.empty()) {
                return false; // няма отваряща скоба за тази
            }
```

```
 char otvarqshta = s.top(); // взимаме последната отворена
                s.pop(); // махаме я от стека

                // Проверяваме дали отговаря
                if ((znak == ')' && otvarqshta != '(') ||
                    (znak == ']' && otvarqshta != '[') ||
                    (znak == '}' && otvarqshta != '{')) {
                return false; // грешен ред на скобите
                }
            }
        }

    // Ако няма останали скоби в стека - всичко е наред
    return s.empty();
}
```

```
int main() {
    string vhod;

    cout << "Vavedi skobi: ";
    cin >> vhod;

    if (pravilniSkobi(vhod)) {
        cout << "Skobite sa pravilni!" <<
endl;
    } else {
        cout << "Skobite NE sa pravilni!" <<
endl;
    }
     return 0; }
```

ПРЕОБРАЗУВАНЕ НА АРИТМЕТИЧНИ ИЗРЗИ В ПОСТФИКСЕН ЗАПИС

2 + 3  23+

2 + 4 * 5  245*+

(7−(1+3))×4  713+-4*

Идея на алгоритъма

1.Имаме **стек** –винаги работим с най-горното.

2.Четем израза символ по символ.

3.Ако символът е **число** → слагаме го в стека.

4.Ако символът е **оператор** (+, -, *, /):

- взимаме двете най-горни числа от стека
- смятаме резултата
- връщаме резултата обратно в стека.

5.Накрая в стека остава **само един елемент** – това е отговорът.

Стъп-ка	Сим-вол	Действие 713+-4*	Съдържание на стека
1	7	Число → слагаме в стека	7
2	1	Число → слагаме в стека	7 1
3	3	Число → слагаме в стека	7 1 3
4	+	Оператор → взимаме горните две: 1 и 3, смятаме 1+3=4, връщаме в стека	7 4
5	-	Оператор → взимаме горните две: 7 и 4, смятаме 7-4=3, връщаме в стека	3
6	4	Число → слагаме в стека	3 4
7	*	Оператор → взимаме 3 и 4, смятаме 3*4=12, връщаме в стека	12

Резултат:12

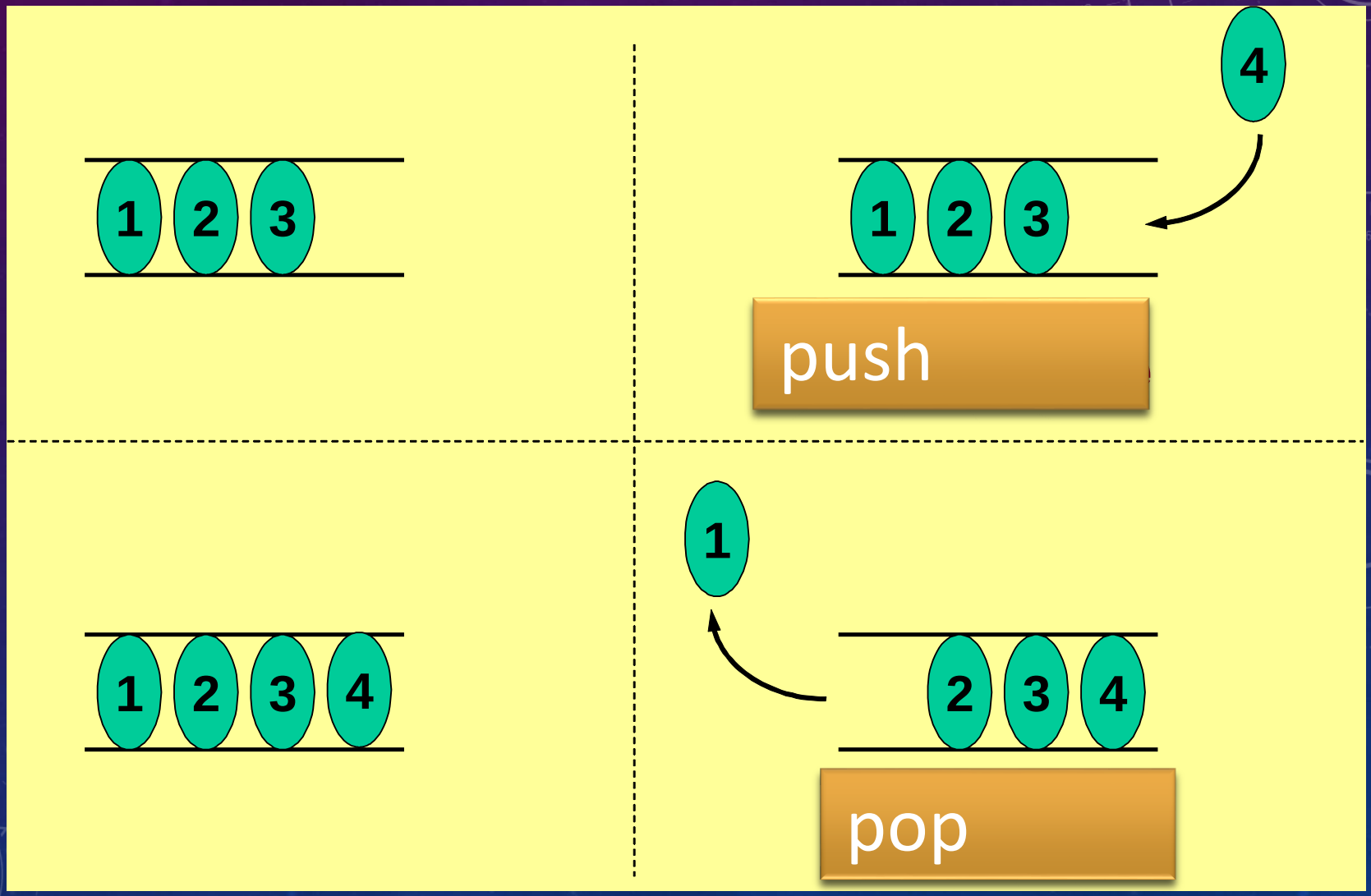
```
#include <iostream>
#include <stack>
#include <string>
using namespace std;
int main() {
    string p;
    cin >> p; // четем постфиксния израз без интервали,
например "23+"
    stack<int> s; // създаваме празен стек
    for (int i = 0; i < p.length(); i++)
    {
        char c = p[i];
        // Ако е цифра
        if (c >= '0' && c <= '9') {
            int a = c - '0'; // превръщаме символ в число
            s.push(a);
        }
        // Ако е оператор
        else {
            int x = s.top(); s.pop();
            int y = s.top(); s.pop();
            int res;
```

```
 switch (c) {
        case '+': res = y + x; break;
        case '-': res = y - x; break;
        case '*': res = y * x; break;
        case '/': res = y / x; break;
    default: cout << "Nevaliden operator!\n";
return 1;
            }
            s.push(res);
        }
    }
    // Резултатът е най-горе в стека
    if (!s.empty()) {
        cout << "Rezultat: " << s.top() <<
endl; } return 0; }
```

5.ОПАШКА

- Опашката е абстрактен тип данни, който съхранява елементи от произволен тип.
- Вмъкването и изтриването на елементи в опашка се извършва по схемата “First In First Out” (FIFO)- - първият елемент в опашката винаги ще бъде обработен най-напред.
- Директен достъп има само до елементите в *началото(front)* и *края(rear)* на опашката.

ИЛЮСТРАЦИЯ НА ОПАШКА



ФУНКЦИИ ОТ СТАНДАРТНАТА БИБЛИОТЕКА

#include <queue>	
queue <typename> q	q – опашка от елементи от тип typename
q.push(x)	включване на елемент x в края на опашката
q.pop()	премахване на елемента от началото на опашката
q.front()	достъп до елемента в началото на опашката
q.back()	достъп до елемента в края на опашката
q.empty()	проверка дали опашката е празна
q.size()	връща броя на елементите в опашката

УПРАЖНЕНИЕ

Задача 1. Да се промени стойността на елемент, намиращ се:

а) в началото на опашката

б) накрая на опашка

Задача 2. В празна опашка последователно постъпват елементите 2, 5, 7, 12, 1, 10:

а) Начертайте схема, отразяваща разположението на елементите в опашката.

б) До кои елементи от опашката има достъп?

в) До кои елементи от опашката има достъп след отстраняването на два елемента?

г) До кои елементи от опашката има достъп след отстраняването на един елемент, постъпването на елементите 4, 8 и отстраняването на четири елемента. Подкрепете отговора със схемата.

Задача 3.

От клавиатурата се въвеждат символите H, E, L, L, O, които постъпват последователно в опашка. Определете думата, която се получава чрез последователното изваждане на символите от опашката.

Задача 4. В магазин влизат N на брой хора, като се въвеждат по имена. Да се напише програма, която извежда имената на хората по реда по куйто ще бъдат обслужени.

```
int main() {
    int N;
    cout << "Въведете брой хора: ";
    cin >> N;
    queue<string> opashka; // Опашка за имената
    // Въвеждаме имената и ги добавяме в опашката
    for (int i = 0; i < N; i++) {
        string ime;
   cout << "Въведете име на човек " << i + 1 << ": ";
        cin >> ime;
        opashka.push(ime);
    }
    // Извеждаме реда на обслужване
    cout << "\nРед на обслужване:\n";
    while (!opashka.empty()) {
        cout << opashka.front() << endl; // Показваме първия
        opashka.pop();                   // Премахваме го (обслужен)
    }

    return 0;
}
```

6.ПРИОРИТЕТНА ОПАШКА

Както и обикновената опашка, приоритетната опашка има две основни операции добавяне и премахване на елемент. За разлика от опашката критерият за напускането е не реда на постъпване на елемента, а неговия приоритет.

ФУНКЦИИ ОТ СТАНДАРТНТА БИБЛИОТЕКА

`#include <queue>`	
`priority_queue <typename> pq`	pq – приоритетна опашка с елементи от тип typename
`pq.push(x)`	включване на елемент x опашката
`pq.pop()`	премахване на елемента с най-висок приоритет
`pq.top()`	достъп до елемента с най-висок приоритет
`pq.empty()`	проверка дали опашката е празна
`pq.size()`	връща броя на елементите в опашката

Задача 6. Напишете програма, която чете **N** цели числа – стойностите на съкровища.
След това изведете числата в низходящ ред (от най-скъпото съкровище към най-евтиното).

Вход:

- На първия ред – едно цяло число **N** (брой съкровища).
- На втория ред – N цели числа, разделени с интервал.

Изход:

- Числата, подредени от най-голямото към най-малкото, разделени с интервал.

Пример:

Вход:

```
5
10 50 20 5 40
```

Изход:

```
50 40 20 10 5
```

Обяснение на решението

1. **Четене на броя елементи**
2. **Създаване на приоритетна опашка**

priority_queue<int> pq;

priority_queue е специален тип опашка, която винаги държи **най-големия** елемент най-отгоре.

- При добавяне (push) тя автоматично подрежда елементите.
- При вземане (top) винаги получаваме най-големия.

3. Добавяне на всички числа в приоритетната опашка

След всяко pq.push(x) опашката сама се грижи за реда на елементите

4. **Извеждане в низходящ ред**

pq.top() връща най-големия елемент в момента.

pq.pop() го премахва.

Процесът се повтаря, докато опашката се изпразни.

```cpp
#include <iostream>
#include <queue>   // за priority_queue
using namespace std;

int main() {
    int n;
    cin >> n;
// 1) четем броя на съкровищата
// 2) правим приоритетна опашка (най-голямото излиза първо)
    priority_queue<int> pq;
//3)четем стойностите и ги добавяме
    for (int i = 0; i < n; i++) {
        int x;
        cin >> x;
        pq.push(x);
    }

// 4) печатаме от най-скъпо към най-евтино
    bool first = true;
    while (!pq.empty()) {
        if (!first) cout << " ";
        cout << pq.top();   // най-голямото число в момента
        pq.pop();
// махаме го
        first = false;
    }
    cout << endl;
    return 0;
}
```