Деструктори, динамично заделяне на памет, стек.

гл.ас., д-р. Нора Ангелова

ДЕСТРУКТОР

- Носи името на класа
- Пред името стои знакът ~
- Извиква се при разрушаване на обекти
  - Разрушаване на обект чрез оператора delete
  - Излизане от блок, в който е бил създаден обект на класа
- Извикват се в обратен ред на конструкторите

```
~rat() {
   //…
}
```

Ако конструкторът или някоя член-функция реализира динамично заделяне на памет за някоя член-данна, използването на деструктор е задължително, тъй като в този случай той трябва да освободи заетата памет.

ПРИМЕР

- Излизане от блок, в който е бил създаден обект на класа

```
class B {
  private:
    int b;
  public:
    B(int bData = 1){
        b = bData;
        cout << "B(" << bData << ")" << endl;
    }

    ~B() {
        cout << "~B" << b << endl;
    }
};

void print () {
  B test1;
}

int main() {
    print();
    return 0;
}
```

Резултат:
B(1)
~B1

Пример

- Разрушаване на обект чрез оператора delete

```
class B {
  private:
    int b;
  public:
    B(int bData = 1){
        b = bData;
        cout << "B(" << bData << ")" << endl;
    }

    ~B() {
        cout << "~B" << b << endl;
    }
};

int main() {
    B * pointer = new B;
    delete pointer;

    return 0;
}
```

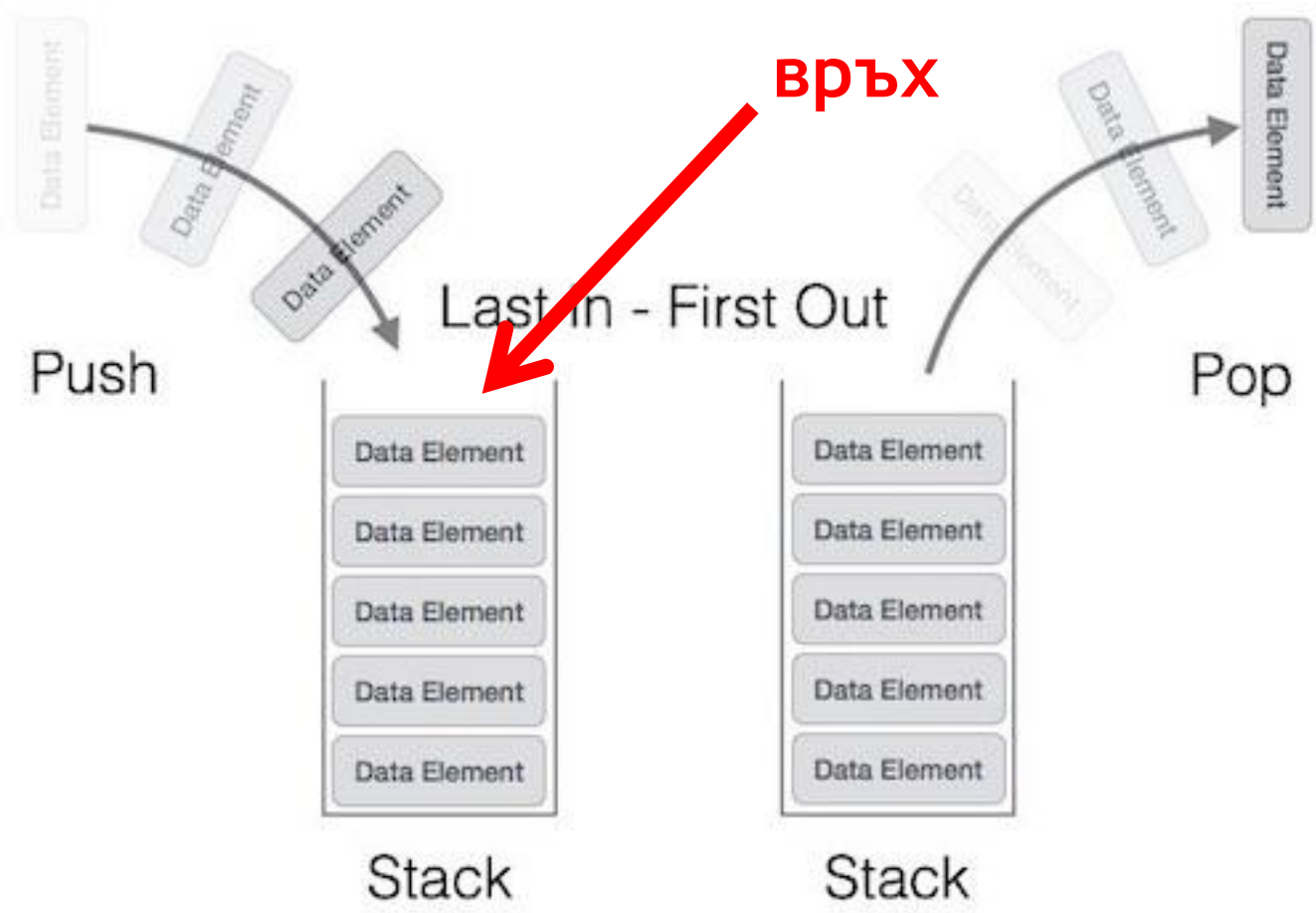
Резултат:
B(1)
~B1

Създаване и разрушаване на масиви от обекти

```
int rows = 3;
int cols = 5;
int ** matrix = new int* [rows];
assert(matrix != NULL);

for (int i=0; i<rows; i++) {
   matrix[i] = new int [cols];
   assert(matrix[i] != NULL);
}

for (int i=0; i<rows; i++)
  for (int j=0; j<cols; j++)
    cin >> matrix[i][j];

for (int i=0; i<rows; i++) {
  for (int j=0; j<cols; j++)
    cout << matrix[i][j] << " ";
  cout << endl;
}

for (int i=0; i<rows; i++)
  delete [] matrix[i];

delete [] matrix;
```

СТЕК

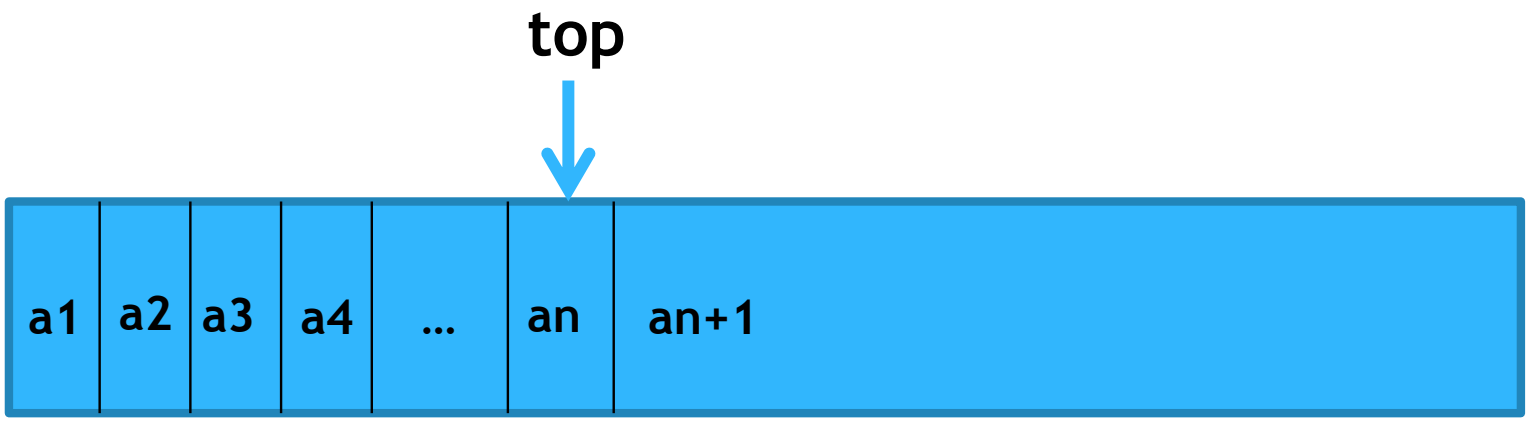
- Хомогенна линейна структура от данни
- „последен влязъл - пръв излязъл“ (LIFO)



СТЕК

Последователно представяне

- Масив
- push - включване на елемент
- pop - изключване на елемент



СТЕК

```
const int NUM = 100;
class stack {
public:
  stack(); // конструктор по подразбиране
  void push(int); // включва елемент в стек
  void pop(int&); // изключва елемент от стек
  bool empty() const; // проверява дали е празен
  bool full() const; // проверява дали е пълен
private:
  int top; // указател към върха на стека
  int arr[NUM]; // представяне на стека
};
```

СТЕК

```
stack::stack() {
    top = -1;
}

void stack::push(int x) {
    if (!full()) {
        top++;
        arr[top] = x;
    }
    else {
        cout << "The stack is full.\n";
        exit(1);
    }
}
```

СТЕК

```
void stack::pop(int& x) {
    if (!empty()) {
        x = arr[top];
        top--;
    } else {
        cout << "The stack is empty.\n";
        exit(1);
    }
}

bool stack::empty() const
{
    return top == -1;
}
```

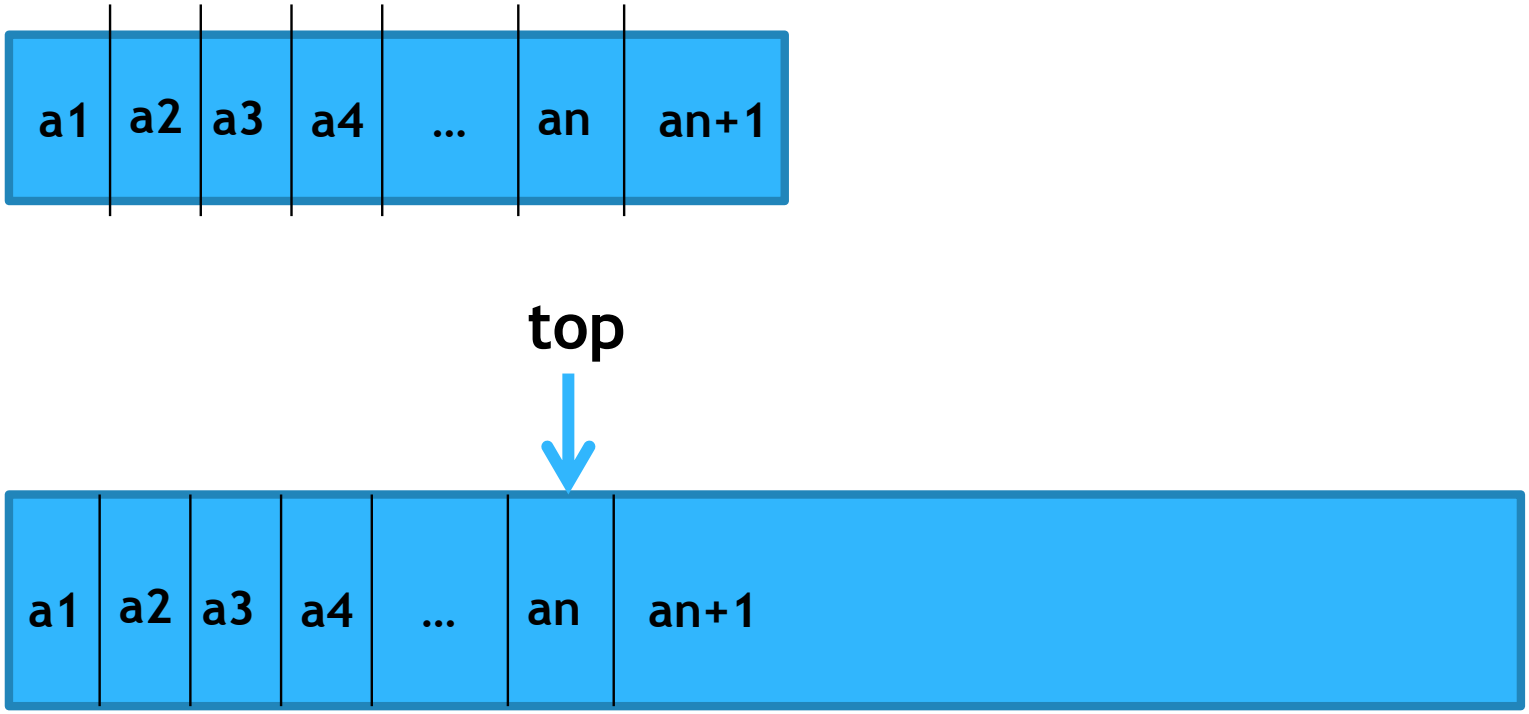
Стек

```
bool stack::full() const {
    return top == NUM-1;
}

void stack::print() {
    int x;
    while (!empty()) {
        pop(x);
        cout << x << " ";
    }
    cout << endl;
}
```

СТЕК

Последователно представяне

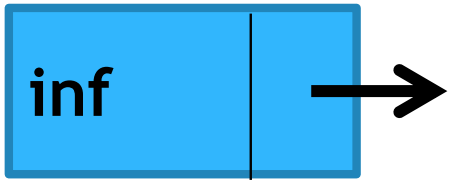
- Динамичен масив



СТЕК

Свързано представяне

```
struct item
{
    int inf;
    item * link;
};
```



КРАЙ