# CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

CHƯƠNG 4

# DANH SÁCH LIÊN KẾT

(Linked list)

GV: Bùi Văn Thượng
Email: thuong.bui@ut.edu.vn

# NỘI DUNG CHÍNH

- GIỚI THIỆU TỔNG QUAN
- DANH SÁCH LIÊN KẾT ĐƠN
- NGĂN XẾP – STACK
- HÀNG ĐỢI QUEUE
- MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

# 1 GIỚI THIỆU TỔNG QUAN

## Biến tĩnh và biến động

| | |
|---|---|
| **Biến tĩnh** | Là biến được khai báo bằng cú pháp khai báo biến, có tên và được cấp phát một vùng nhớ cố định trước khi sử dụng. Vùng nhớ cố định đó luôn tồn tại xuyên suốt thời gian thực thi chương trình, bao gồm cả những biến mà chương trình chỉ sử dụng 1 lần rồi bỏ. |
| **Biến động** | Là biến được tạo ra và khởi tạo giá trị khi chương trình hoạt động. Biến động có thể được thu hồi bộ nhớ ngay khi có lệnh yêu cầu giải phóng vùng nhớ nó đang chiếm giữ để có thể sử dụng vào việc khác. |

# 1 GIỚI THIỆU TỔNG QUAN

## Một số đặc điểm của biến động

Biến động không được khai báo tường minh, không có tên gọi.

Biến động có thể xin khi cần, giải phóng khi sử dụng xong.

Biến động linh động về kích thước cho nên tận dụng hiệu quả bộ nhớ.

# 1 GIỚI THIỆU TỔNG QUAN

## Cấp phát bộ nhớ động

Trong lập trình, đặc biệt với kiểu dữ liệu mảng, việc khai báo số lượng phần tử cố định từ trước thường gây ra tình trạng lãng phí hoặc thiếu bộ nhớ. Để khắc phục, C++ cho phép **cấp phát bộ nhớ động** bằng toán tử **new**, giúp cấp phát bộ nhớ theo nhu cầu.

Tuy nhiên, việc cấp phát động khiến dữ liệu có thể **nằm rải rác** trong bộ nhớ. Để quản lý các dữ liệu này theo một danh sách có trật tự, mỗi phần tử của danh sách cần chứa địa chỉ của phần tử tiếp theo hoặc trước đó.

# 1 GIỚI THIỆU TỔNG QUAN

## Cấu trúc tự trỏ

**Cấu trúc tự trỏ** là một kiểu cấu trúc dữ liệu mà trong đó, ngoài các thành phần chứa thông tin của chính nó, còn có chứa ít nhất một con trỏ trỏ đến một vùng dữ liệu có kiểu chính là kiểu của cấu trúc đó. Điều này cho phép các phần tử của một danh sách có thể liên kết với nhau trong bộ nhớ dù chúng không nằm liền kề, tạo thành các cấu trúc dữ liệu linh hoạt như danh sách liên kết.

Ví dụ về cấu trúc tự trỏ

```
struct Student {
        char name[33];
        int code;
        Student *next;
}
```

# 1 GIỚI THIỆU TỔNG QUAN

## Cấu trúc dữ liệu dạng danh sách

Danh sách là một kiểu dữ liệu dạng tuyến tính, nó bao gồm tập hợp các phần tử có cùng kiểu dữ liệu. Mỗi phần tử trong danh sách có nhiều nhất 1 phần tử đứng trước và nhiều nhất 1 phần tử đứng sau.

Có hai hình thức tổ chức cơ bản của danh sách là mảng (liên kết ngầm) và danh sách liên kết (liên kết tường minh).

# 1 GIỚI THIỆU TỔNG QUAN

## Cấu trúc dữ liệu dạng danh sách

### 1) **Mảng** – Array

Phải lưu trữ liên tiếp các phần tử trong bộ nhớ trong đó:

- *A[i] là phần tử thứ i+1 trong danh sách.*
- *A[i] và A[i+1] là các phần tử kế cận trong danh sách.*

Ưu điểm: *Truy xuất trực tiếp, nhanh chóng.*

Nhược điểm:

- *Sử dụng bộ nhớ kém hiệu quả.*
- *Kích thước cố định.*
- *Các thao tác thêm vào, loại bỏ không hiệu quả.*

# 1 GIỚI THIỆU TỔNG QUAN

## Cấu trúc dữ liệu dạng danh sách

### 2) Danh sách liên kết (Tiếng Anh - Linked List)

Cấu trúc dữ liệu cho mỗi phần tử là 1 cấu trúc tự trỏ bao gồm:

- *Thông tin bản thân phần tử.*
- *Địa chỉ của phần tử kế trong danh sách.*

Mỗi phần tử của danh sách liên kết là một biến động.

Ưu điểm: *Sử dụng hiệu quả bộ nhớ và linh động về số lượng phần tử.*

Các dạng của danh sách liên kết bao gồm:

- *Danh sách liên kết đơn,*
- *Danh sách liên kết kép*
- *Danh sách liên kết vòng.*

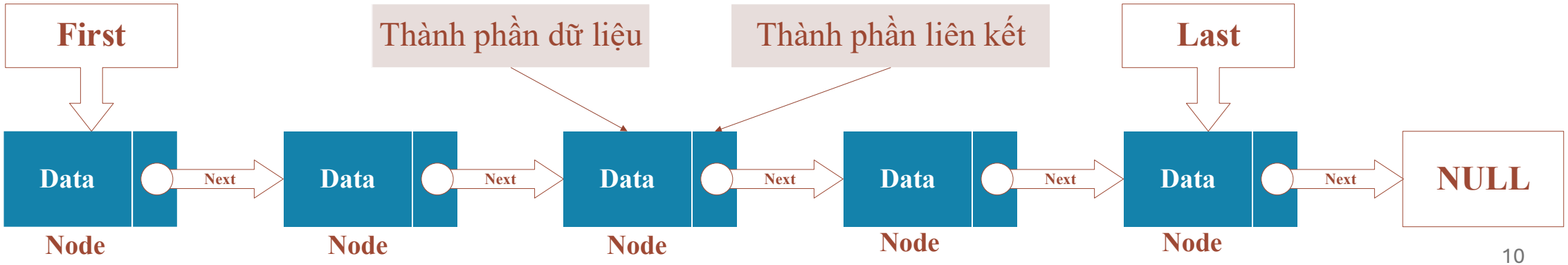
# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Cấu trúc dữ liệu của danh sách liên kết đơn

Danh sách liên kết đơn (Singly Linked List) là DSLK mà mỗi phần tử trong danh sách sẽ liên kết với phần tử liền sau nó. Mỗi phần tử trong danh sách liên kết đơn là một cấu trúc có hai thành phần:

- **Thành phần dữ liệu**: Lưu trữ thông tin về bản thân phần tử.
- **Thành phần liên kết**: Lưu địa chỉ phần tử đứng sau trong danh sách hoặc bằng NULL nếu là phần tử cuối danh sách.

Cấu trúc dữ liệu của DSLK đơn bao gồm 2 con trỏ lần lượt lưu địa chỉ của phần tử (hoặc nút - node) đầu tiên và phần tử cuối cùng trong danh sách.



# 2 DANH SÁCH LIÊN KẾT ĐƠN

**Ví dụ**. Biểu diễn danh sách sinh viên bằng DSLK đơn

```
//Cấu trúc của nút
struct Student {
        char name[33];
        int code;
        Student *next;
};
//Cấu trúc của danh sách
struct List {
        Student *first;
        Student *last;
};
```

# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết  đơn

### 1. Khởi tạo danh sách liên kết đơn rỗng

Địa chỉ của nút đầu tiên và nút cuối cùng không có.

```
void createList(List &ListStudent) {
        ListStudent.first = NULL;
        ListStudent.last = NULL;
}
```

# 2 DANH SÁCH LIÊN KẾT ĐƠN

**Các giải thuật trên danh sách liên kết  đơn**

## 2. Tạo một phần tử mới

Để tạo phần tử mới ta thực hiện theo các bước sau đây:

- *Dùng toán tử **new** xin cấp phát một vùng nhớ cho phần tử mới của DSLK.*
- *Nhập thông tin cho phần tử mới. Con trỏ **next** được đặt bằng NULL.*
- *Gắn phần tử vừa tạo vào danh sách.*

**Để gắn phần tử mới tạo vào DSLK ta có 3 cách thêm như sau**:

- *Thêm phần tử vào đầu DSLK*
- *Thêm phần tử vào cuối DSLK*
- *Thêm phần tử vào sau một phần tử của DSLK*

# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết đơn

### 2. Tạo một phần tử mới

**Ví dụ**: Hàm tạo nút trong DSLK đơn sinh viên

```
Student* createNode() {
        Student *x = new Student[1];
        cout<<"Enter student code: ";
        cin>> x->code;
        cout<<"Enter student name: ";
        cin.ignore(1);
        cin.getline(x->name, 33);
        x->next = NULL;
        return x;
}
```

# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết đơn

### 3. Thêm phần tử vào đầu danh sách

**Các bước thực hiện**:

- *Tạo và cấp phát bộ nhớ cho 1 nút mới.*
- *Nếu DSLK rỗng thì gán phần tử đầu và cuối của DSLK bằng chính nút mới tạo.*
- *Nếu DSLK khác rỗng thì cho con trỏ **next** của nút mới trỏ đến phần tử đầu tiên của DSLK sau đó cho con trỏ đầu của DSLK trỏ vào nút mới tạo.*

# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết đơn

### 3. Thêm phần tử vào đầu danh sách

**Ví dụ minh họa:**

```
void insertFirst (List &ListStudent, Student *x) {
        if (ListStudent.first == NULL) {
                ListStudent.first = x;
                ListStudent.last = x;
        } else {
                x->next = ListStudent.first;
                ListStudent.first = x;
        }
}
```

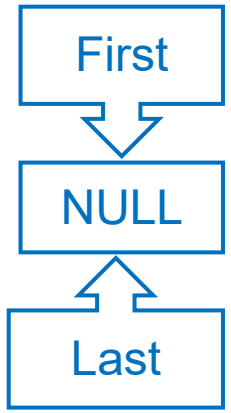
# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết đơn

### 3. Thêm phần tử vào đầu danh sách

**Trường hợp DSLK rỗng:**


Node mới X



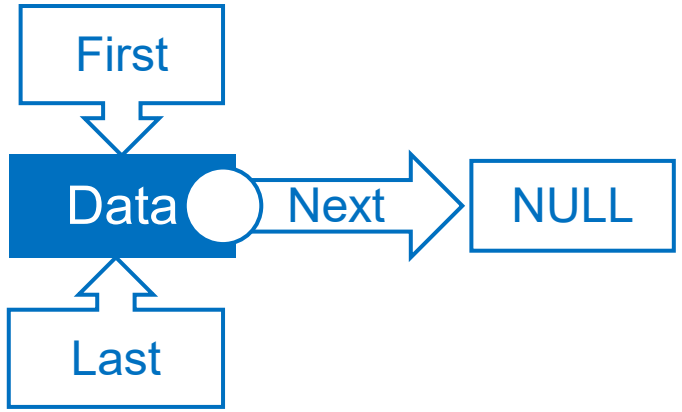

Danh sách liên kết

# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết đơn

### 3. Thêm phần tử vào đầu danh sách

**Trường hợp DSLK rỗng:**




Danh sách liên kết

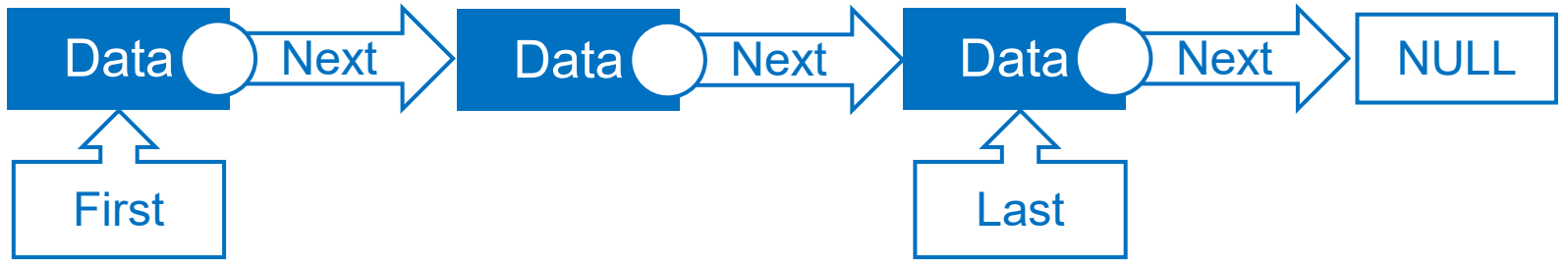
# 2 DANH SÁCH LIÊN KẾT ĐƠN

**Các giải thuật trên danh sách liên kết đơn**

## 3. Thêm phần tử vào đầu danh sách

**Trường hợp DSLK Khác rỗng:**


Node mới X




Danh sách liên kết

# 2 DANH SÁCH LIÊN KẾT ĐƠN

**Các giải thuật trên danh sách liên kết  đơn**

## 3. Thêm phần tử vào đầu danh sách

**Trường hợp DSLK Khác rỗng:**


Danh sách liên kết

# 2 DANH SÁCH LIÊN KẾT ĐƠN

**Các giải thuật trên danh sách liên kết  đơn**

## 3. Thêm phần tử vào đầu danh sách

**Trường hợp DSLK Khác rỗng:**


Danh sách liên kết

# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết đơn

### 4. Thêm phần tử vào cuối danh sách

**Các bước thực hiện:**

- Tạo và cấp phát bộ nhớ cho 1 nút mới.
- Nếu DSLK rỗng thì gán phần tử đầu và cuối của DSLK bằng nút mới tạo.
- Nếu DSLK khác rỗng:
  - *Cho con trỏ **next** của phần tử cuối của DSLK trỏ đến nút mới tạo.*
  - *Gán phần tử cuối của DSLK bằng nút mới tạo và cho con trỏ **next** của nút mới bằng NULL.*

# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết đơn

### 4. Thêm phần tử vào cuối danh sách

**Ví dụ minh họa:**

```
void insertLast (List &ListStudent, Student *x) {
        if (ListStudent.first == NULL) {
                ListStudent.first = x;
                ListStudent.last = x;
        } else {
                ListStudent.last->next = x;
                ListStudent.last = x;
        }
}
```

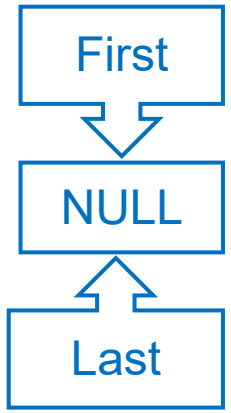
# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết đơn

### 4. Thêm phần tử vào cuối danh sách

**Trường hợp DSLK rỗng:**


Node mới X



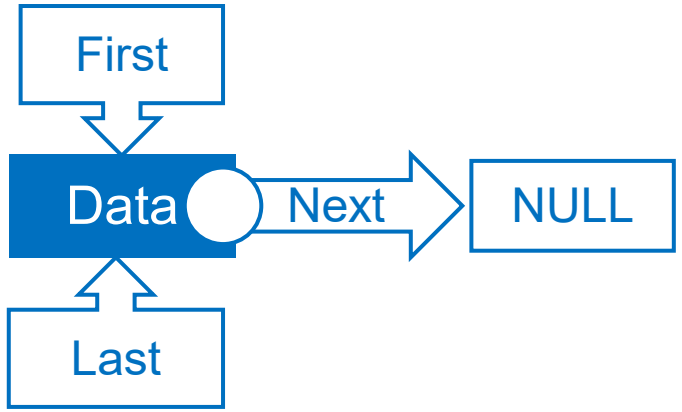

Danh sách liên kết

# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết đơn

### 4. Thêm phần tử vào cuối danh sách

**Trường hợp DSLK rỗng:**




Danh sách liên kết

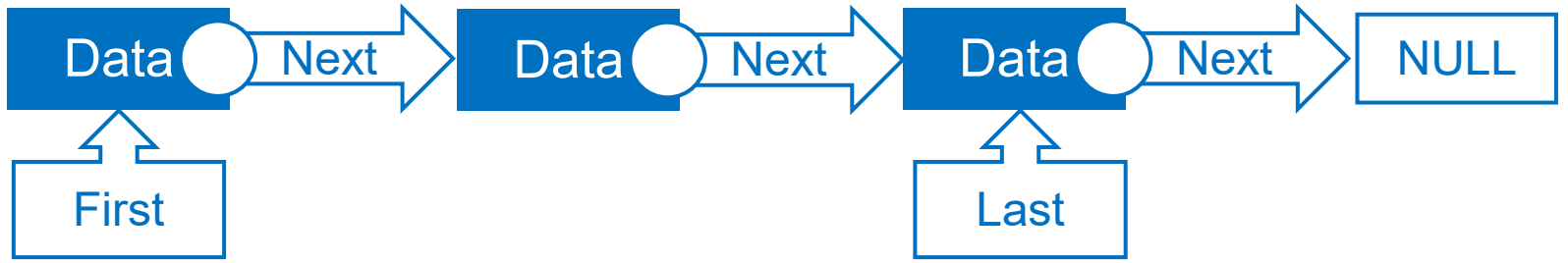
# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết đơn

### 4. Thêm phần tử vào cuối danh sách

**Trường hợp DSLK khác rỗng:**


Node mới X



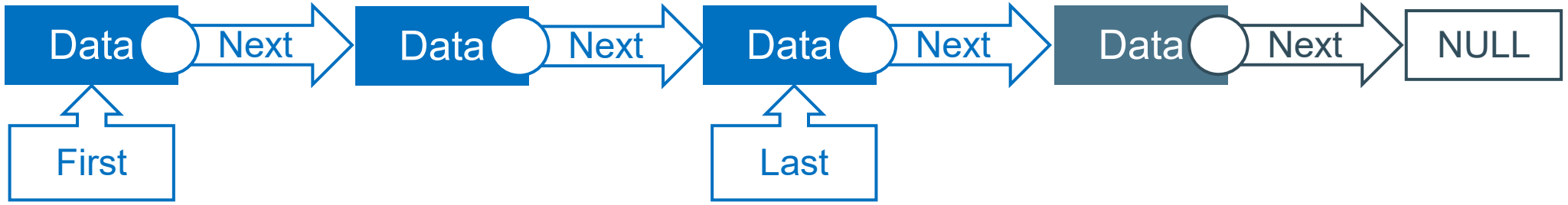

Danh sách liên kết

# 2 DANH SÁCH LIÊN KẾT ĐƠN

**Các giải thuật trên danh sách liên kết đơn**

## 4. Thêm phần tử vào cuối danh sách

**Trường hợp DSLK khác rỗng:**



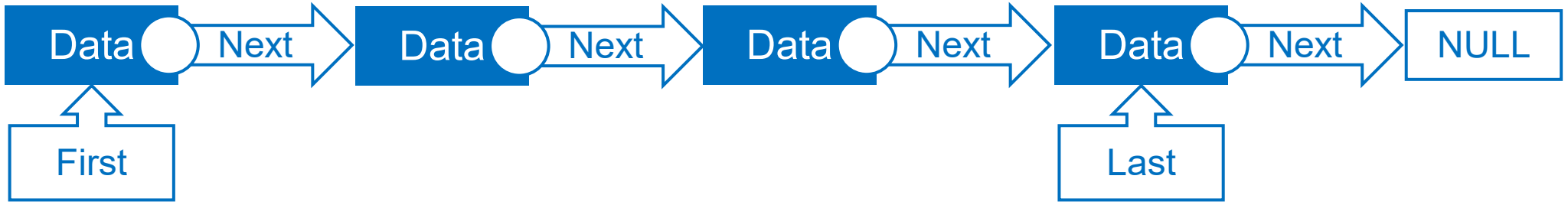

Danh sách liên kết

# 2 DANH SÁCH LIÊN KẾT ĐƠN

**Các giải thuật trên danh sách liên kết đơn**

## 4. Thêm phần tử vào cuối danh sách

**Trường hợp DSLK khác rỗng:**




Danh sách liên kết

# 2 DANH SÁCH LIÊN KẾT ĐƠN

**Các giải thuật trên danh sách liên kết  đơn**

## 5. Thêm phần tử X vào sau nút Q

**Các bước thực hiện:**

- Tạo và cấp phát bộ nhớ cho 1 nút X mới cần thêm.
- Nếu tìm thấy nút Q:
  - *Cho con trỏ next của nút mới X trỏ đến nút kế của Q.*
  - *Cho con trỏ next của Q trỏ vào nút mới.*
  - *Trường hợp Q là nút cuối thì gán phần tử cuối của DSLK bằng nút mới thêm.*
- Nếu không tìm thấy nút Q thì thêm phần tử mới vào đầu hoặc cuối DSLK.

# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết đơn

### 5. Thêm phần tử X vào sau nút Q

**Ví dụ minh họa:**

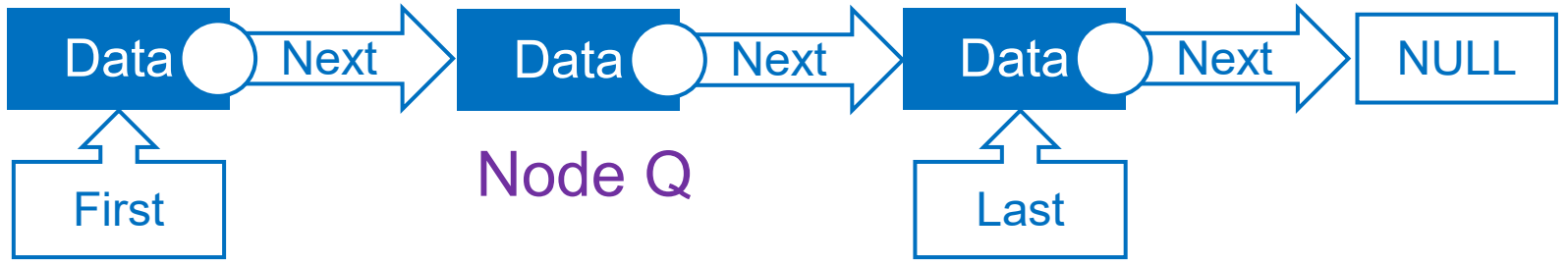
```
void insertAfterNode (List &ListStudent, Student *x, Student *q) {
        if (q!=NULL) {
                x->next=q->next;
                q->next=x;
        if (ListStudent.last==q) {
                ListStudent.last=x;
        }
        } else {
                insertFirst(ListStudent, x); //Thêm X vào đầu danh sách
        }
}
```

# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết đơn

### 5. Thêm phần tử X vào sau nút Q

Node mới X

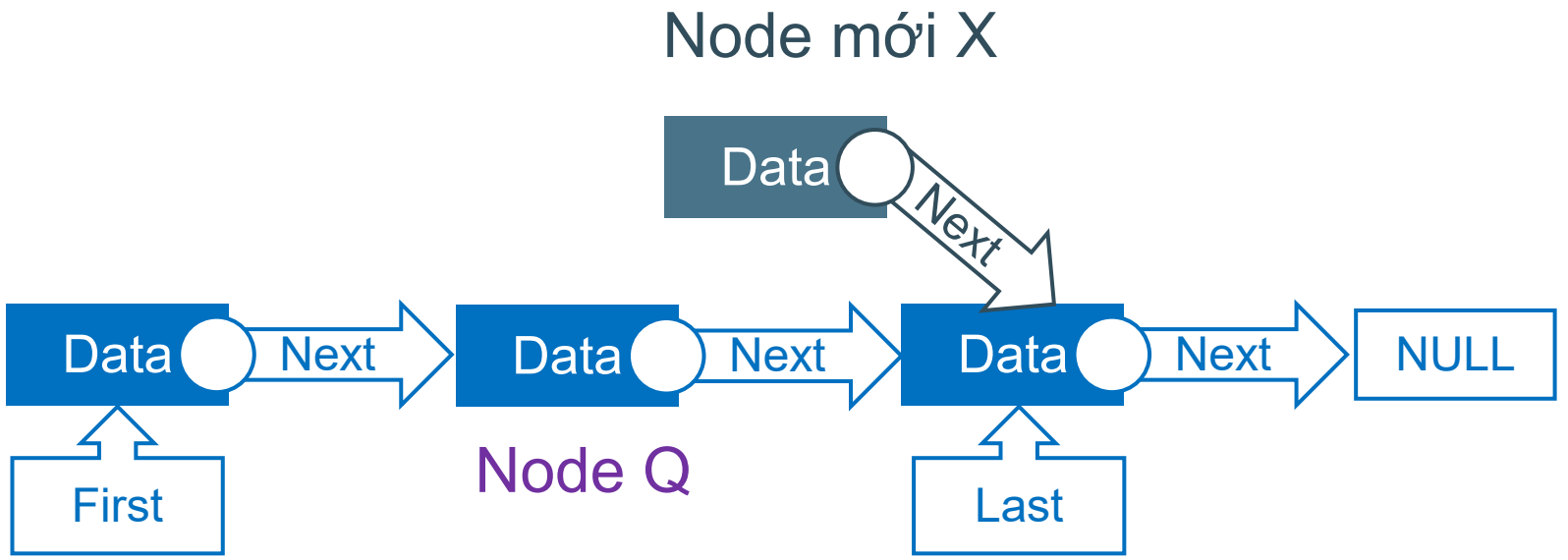




Danh sách liên kết

# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết đơn

### 5. Thêm phần tử X vào sau nút Q

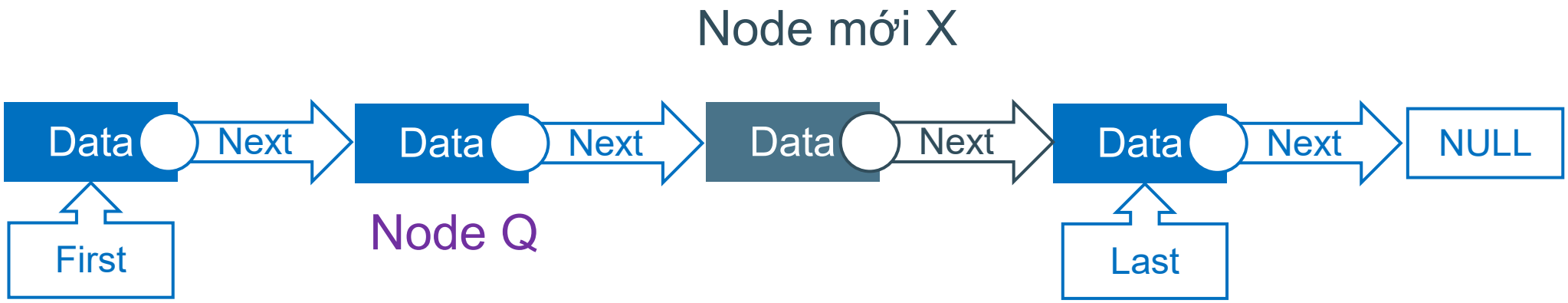



Danh sách liên kết

# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết đơn

### 5. Thêm phần tử X vào sau nút Q

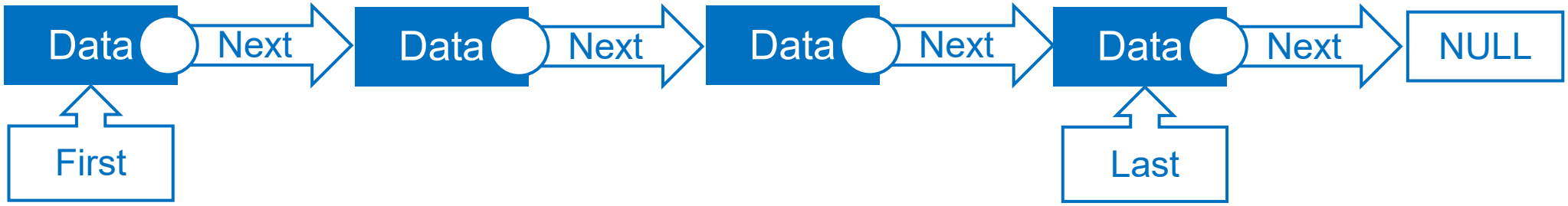



Danh sách liên kết

# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết đơn

### 5. Thêm phần tử X vào sau nút Q




Danh sách liên kết

# 2 DANH SÁCH LIÊN KẾT ĐƠN

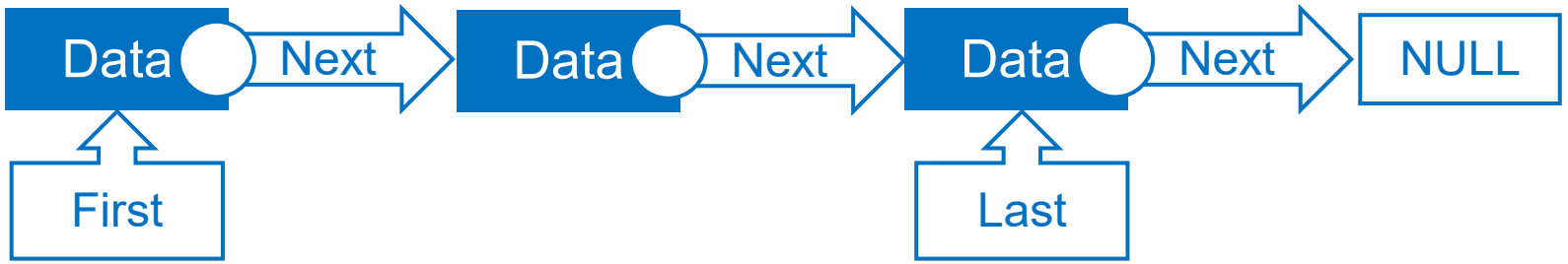
**Các giải thuật trên danh sách liên kết  đơn**

## 5. Thêm phần tử X vào sau nút Q

**Trường hợp nút Q nằm cuối danh sách:**

Node mới X


Node Q



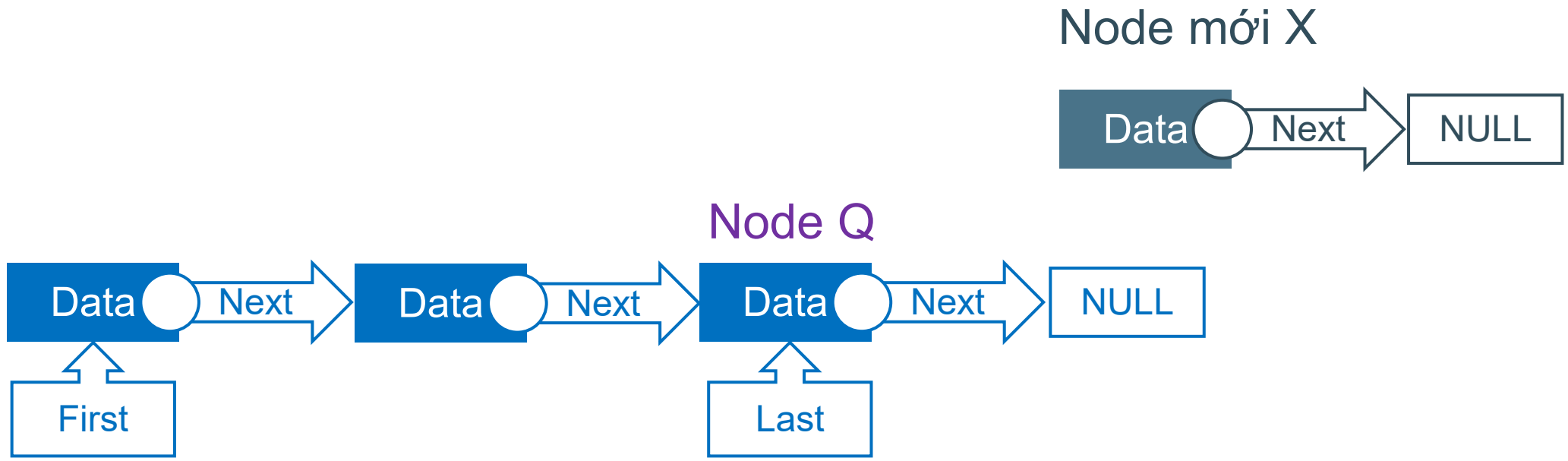

Danh sách liên kết

# 2 DANH SÁCH LIÊN KẾT ĐƠN

**Các giải thuật trên danh sách liên kết đơn**

## 5. Thêm phần tử X vào sau nút Q

**Trường hợp nút Q nằm cuối danh sách:**



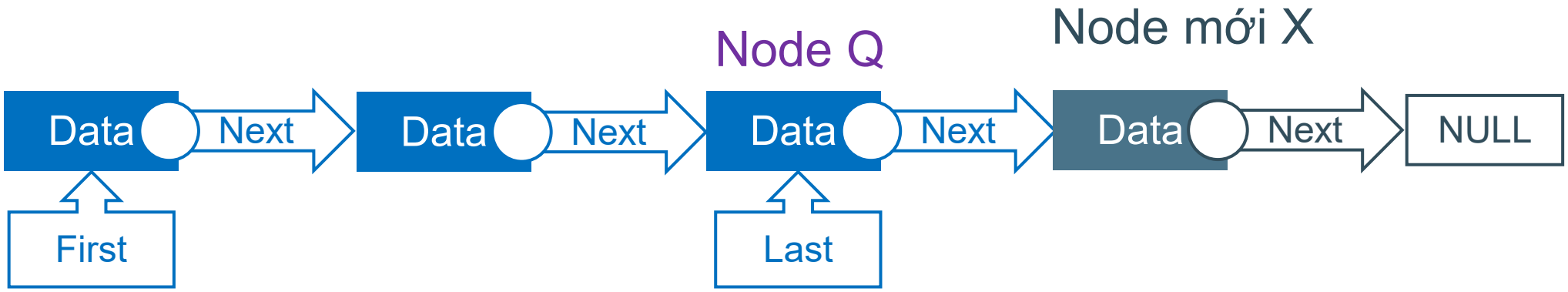

Danh sách liên kết

# 2 DANH SÁCH LIÊN KẾT ĐƠN

**Các giải thuật trên danh sách liên kết đơn**

## 5. Thêm phần tử X vào sau nút Q

**Trường hợp nút Q nằm cuối danh sách:**



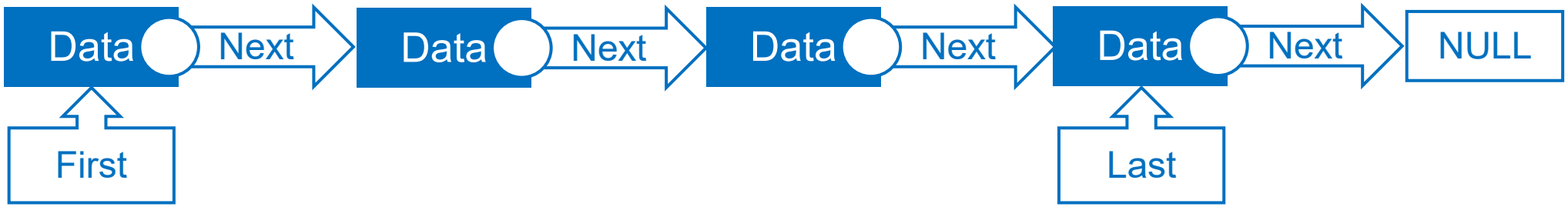

Danh sách liên kết

# 2 DANH SÁCH LIÊN KẾT ĐƠN

**Các giải thuật trên danh sách liên kết  đơn**

## 5. Thêm phần tử X vào sau nút Q

**Trường hợp nút Q nằm cuối danh sách:**




Danh sách liên kết

# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết đơn

### 6. Duyệt danh sách liên kết đơn

Ta sử dụng 1 biến con trỏ tạm dạng nút của DSLK rồi trỏ vào phần tử đầu của DSLK. Từ vị trí này, theo liên kết giữa các nút ta sẽ thực hiện việc duyệt qua từng phần tử trong DSLK.

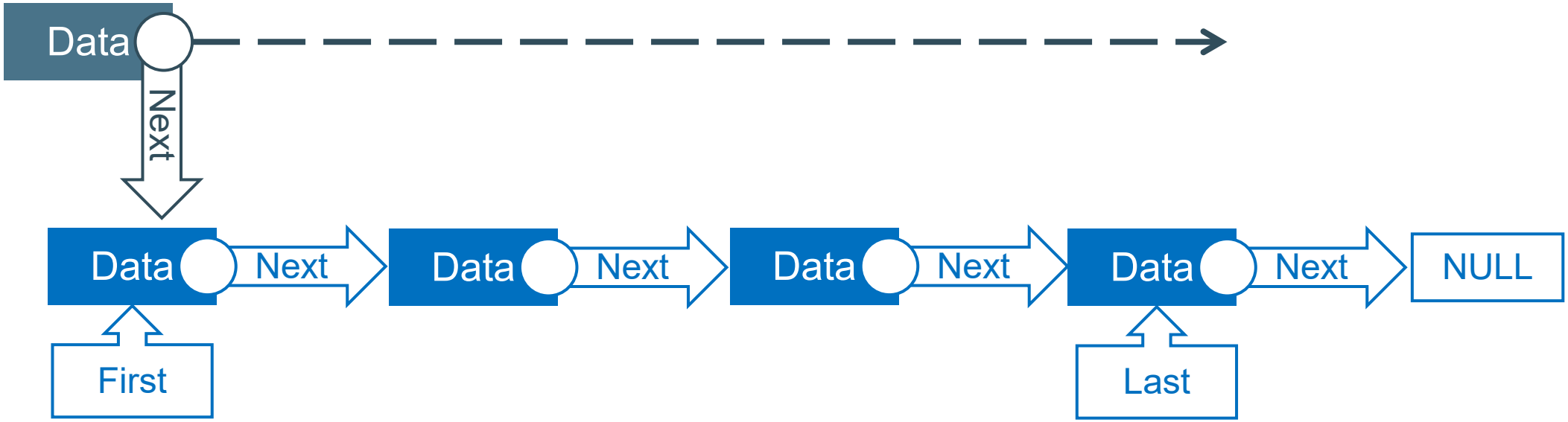
Trong quá trình duyệt, tại mỗi nút ta có thể thực hiện các thao tác như:

- Lấy thông tin phần tử.
- Sửa thông tin phần tử.
- So sánh phần tử.
- Xóa phần tử….

# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết  đơn

### 6. Duyệt danh sách liên kết đơn



# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết đơn

### 6. Duyệt danh sách liên kết đơn

**Hàm minh họa:**

```
void Output (List &ListStudent) {
        Student *temp;
        temp = ListStudent.first;
        cout<<"Danh Sach cac phan tu cua DSLK:\n";
        cout<<"Ma So\tHo Va Ten";
        while (temp != NULL){
                cout<<endl<<temp->code;
                cout<<"\t"<<temp->name;
                temp = temp->next;
        }
}
```

# 2 DANH SÁCH LIÊN KẾT ĐƠN

**Các giải thuật trên danh sách liên kết  đơn**

## 7. Xóa phần tử P khỏi danh sách liên kết đơn

**Các bước thực hiện:**

Tìm nút Q liền trước nút P.

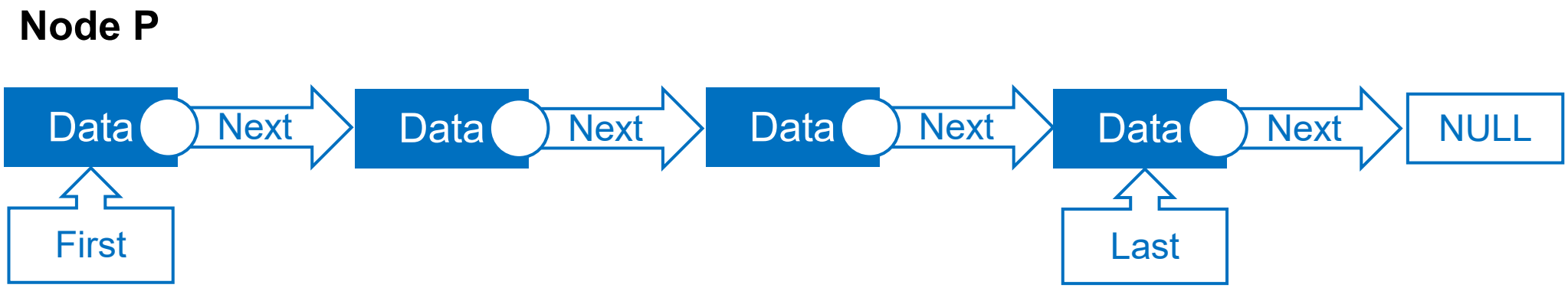
- Nếu tìm thấy Q, cho Q liên kết với nút liền sau của P rồi giải phóng bộ nhớ đã cấp cho P.
- Nếu không tìm thấy Q, tức P là nút đầu tiên, khi đó ta chỉ việc thay đổi nút đầu tiên của DSLK thành nút đứng liền sau P rồi thu hồi bộ nhớ của P.

# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết  đơn

### 7. Xóa phần tử P khỏi danh sách liên kết đơn

**Xóa nút đầu danh sách:**

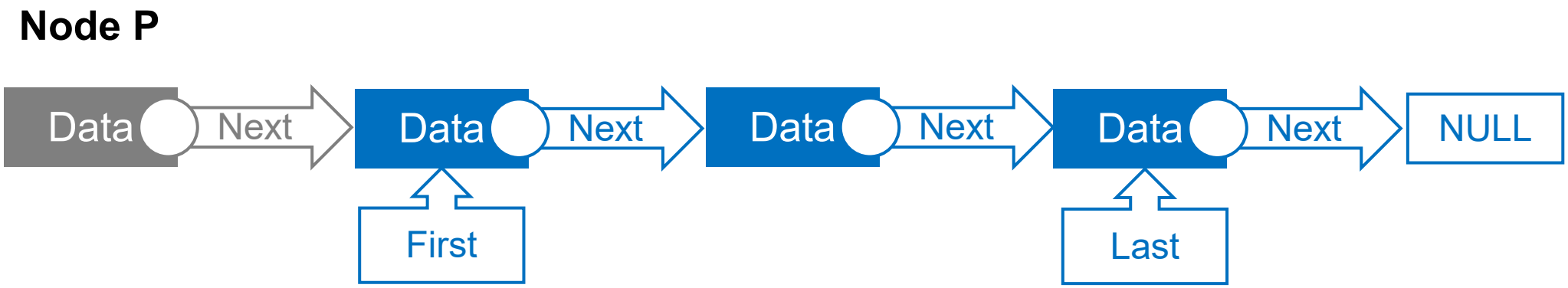


# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết  đơn

### 7. Xóa phần tử P khỏi danh sách liên kết đơn

**Xóa nút đầu danh sách:**

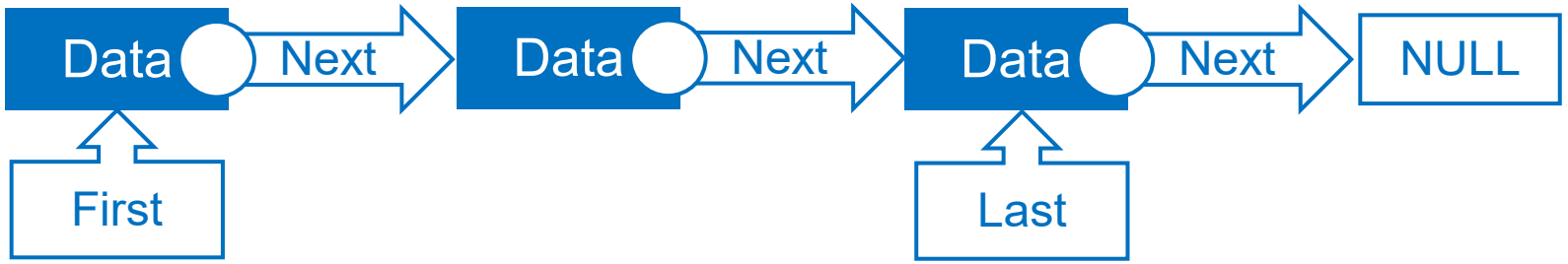


# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết  đơn

### 7. Xóa phần tử P khỏi danh sách liên kết đơn

**Xóa nút đầu danh sách:**

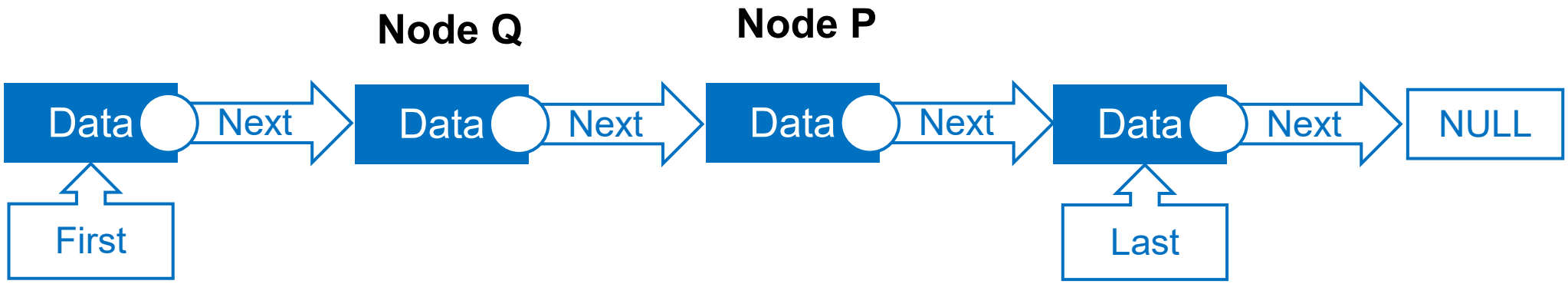


# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết đơn

### 7. Xóa phần tử P khỏi danh sách liên kết đơn

**Xóa nút P đứng sau nút Q:**

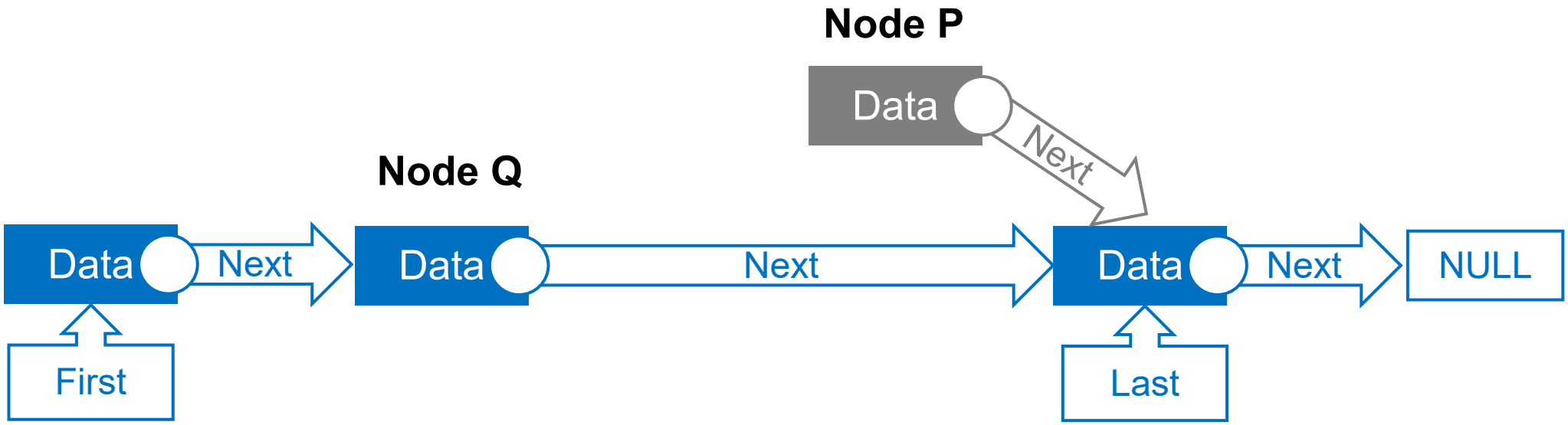


# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết đơn

### 7. Xóa phần tử P khỏi danh sách liên kết đơn

**Xóa nút P đứng sau nút Q:**

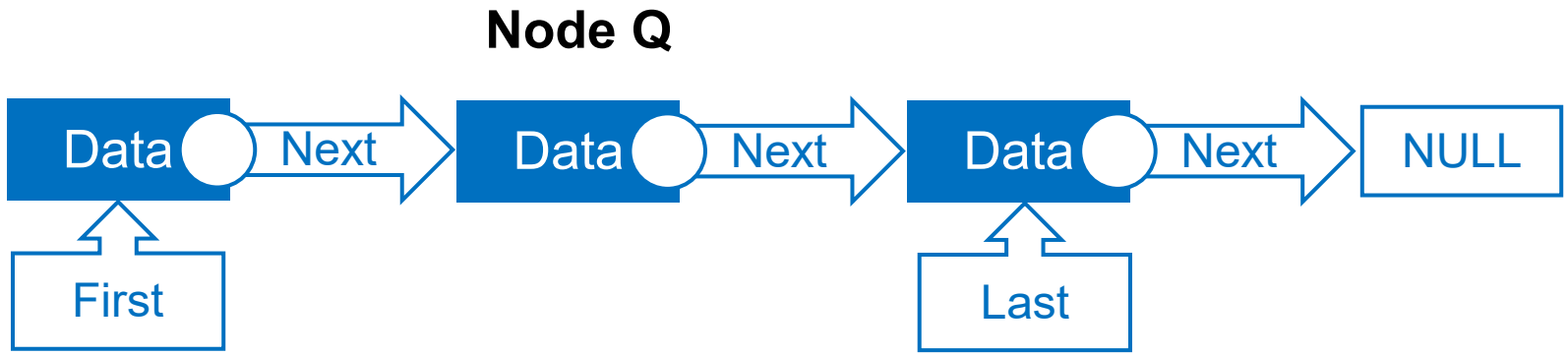


# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết  đơn

### 7. Xóa phần tử P khỏi danh sách liên kết đơn

**Xóa nút P đứng sau nút Q:**



## Các giải thuật trên danh sách liên kết đơn

### 7. Xóa phần tử P khỏi danh sách liên kết đơn

**Hàm minh họa:**

```
bool removeX(List &ListStudent, Student *x){
        Student *p; p = ListStudent.first;
        if (ListStudent.first == x) {
                ListStudent.first = x->next;
                delete x; return true; //Xóa thành công
        }
        while (p != NULL && p ->next != x) p = p->next;
        if (p == NULL) return false; //Không tìm thấy x, xóa thất bại
        else {
                p->next = x->next;
                delete x; return true; //Xóa thành công
        }
}
```

# 2 DANH SÁCH LIÊN KẾT ĐƠN

## Các giải thuật trên danh sách liên kết đơn

### 8. Sắp xếp danh sách liên kết đơn

Có 2 cách tiếp cận:

**Cách 1**: Thay đổi thành phần dữ liệu của các nút.

Ưu điểm: *Cài đặt đơn giản, tương tự như sắp xếp mảng.*

Nhược điểm:

- *Đòi hỏi thêm vùng nhớ khi hoán vị nội dung của 2 phần tử. Điều này chỉ phù hợp với những DSLK có kích thước dữ liệu nhỏ.*
- *Khi kích thước dữ liệu lớn thì chi phí cho việc hoán vị thành phần dữ liệu cũng lớn.*
- *Làm cho thao tác sắp xếp chậm.*

## Các giải thuật trên danh sách liên kết đơn

### 8. Sắp xếp danh sách liên kết đơn

**Cách 2**: Thay đổi thành phần con trỏ **next** trong nút (thay đổi trình tự móc nối của các phần tử sao cho tạo lập nên được thứ tự mong muốn).

Ưu điểm:

- *Kích thước của trường này không thay đổi, do đó không phụ thuộc vào kích thước bản chất dữ liệu lưu tại mỗi nút.*
- *Thao tác sắp xếp nhanh.*

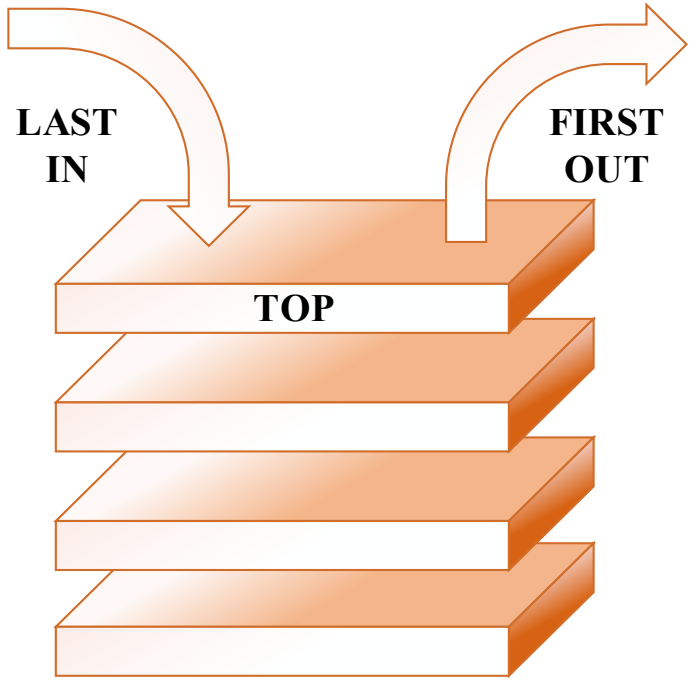
Nhược điểm: *Cài đặt phức tạp.*

***Ghi chú:*** *Các thuật toán sắp xếp DSLK bằng cách thay đổi thành phần liên kết có hiệu quả cao như: Quick Sort, Merge Sort, …*

# 3 NGĂN XẾP - STACK

**Stack** (ngăn xếp): Là một danh sách mà ta giới hạn việc thêm vào hoặc loại bỏ một phần tử chỉ thực hiện tại một đầu của danh sách, đầu này gọi là đỉnh (TOP) của ngăn xếp. Chính vì nguyên tắc này mà ngăn xếp còn được gọi là kiểu dữ liệu có nguyên tắc LIFO (Last In First Out – Vào sau ra trước).

- Các thao tác trên ngăn xếp:
- Thêm đối tượng vào stack.
- Lấy đối tượng từ stack.
- Kiểm tra stack rỗng hay không?
- Lấy giá trị phần tử đỉnh của stack mà không hủy nó.

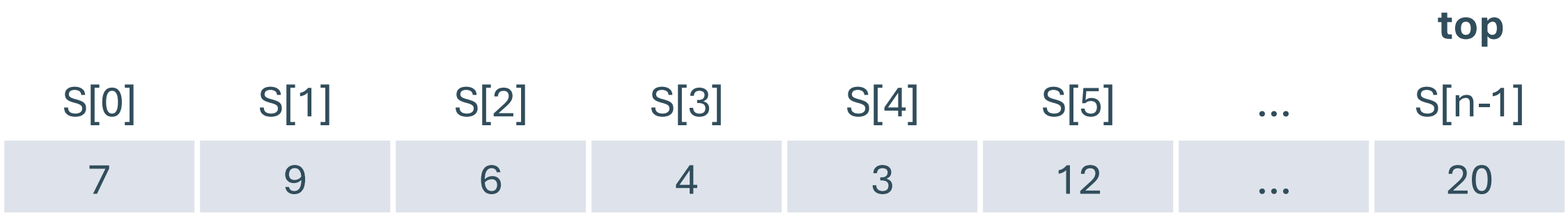


# 3 NGĂN XẾP - STACK

## Cài đặt ngăn xếp bằng mảng một chiều

Để cài đặt ngăn xếp bằng mảng, ta sử dụng mảng một chiều S để biểu biễn ngăn xếp. Thiết lập phần tử đầu tiên của mảng S[0] làm đáy ngăn xếp. Các phần tử tiếp theo được đưa vào ngăn xếp sẽ lần lượt được lưu tại các vị trí S[1], S[2],…

- Nếu hiện tại ngăn xếp có n phần tử thì S[n-1] sẽ là đỉnh của ngăn xếp.
- Để lưu trữ đỉnh hiên tại của ngăn xếp, ta sử dụng một biến top lưu chỉ số của đỉnh (ở đây top = n-1).
- Khi ngăn xếp chưa có phần tử nào ta quy ước top = -1.



# 3 NGĂN XẾP - STACK

```
#include<iostream>
using namespace std;
typedef struct tagStack{ //Cấu trúc Stack
    int S[100];
    int top;
} Stack;
void createStack(Stack &st){
    st.top = -1;
}
bool isEmpty(Stack st){ //Kiểm tra rỗng
    if(st.top == -1) return true;
    else return false;
}
bool isFull(Stack st, int max){
    if(st.top >= max)return true;
    else return false;
}
bool push(Stack &st, int max, int x){
    if(isFull(st, max) == 0){
          st.top++; st.S[st.top] = x;
          return true;
    } else return false;
}
bool pop(Stack &st, int &x){//Lấy phần tử ra
     if(isEmpty(st) == 0){
          x = st.S[st.top]; st.top--;
          return true;
     } else return false;
}
int main()
{
    Stack st;
    int max = 100, x, result, i;
    createStack(st);
    //Tạo Stack chứa các số từ 1 đến 10
    for(i = 1; i <= 10; i++)
          push(st, max, i);
    result = pop(st, x);
    if(result) cout<<"Value get from
Stack:"<<x;
//Kết quả x = 10
}
```

# 3 NGĂN XẾP - STACK

**Ví dụ:** Sử dụng Stack để hiển thị một số nguyên dương ở dạng số nhị phân

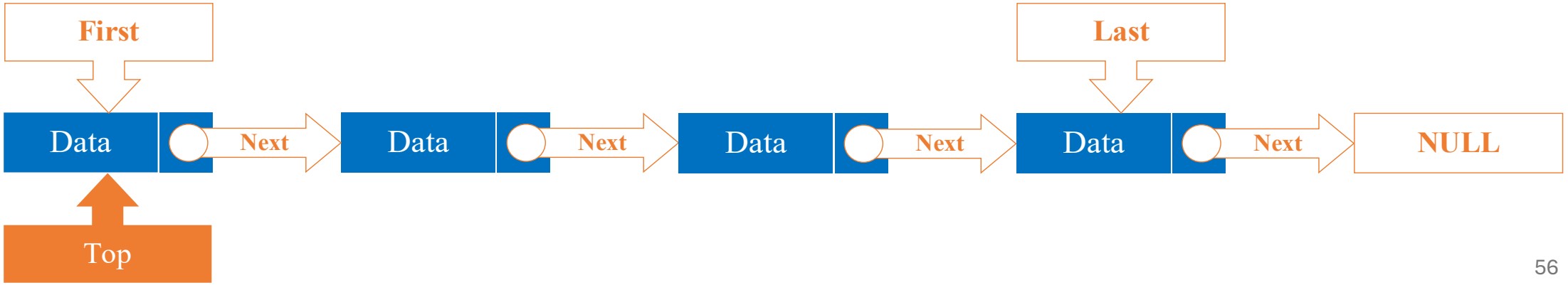
```
#include<iostream>
using namespace std;
struct Stack{
    int a[100], top;
};
int main()
{
    Stack BynaryNumber;
    BynaryNumber.top =- 1;
    int i = 0, x;
    Loop:
    cout<<"Enter positive number: ";
    cin>>x;
    cout<<"Number "<<x<<" in bynary: ";
    if(x <= 0) goto Loop;
    while(x > 0){
        BynaryNumber.a[i]=x%2;
        x /=2; i++;
        BynaryNumber.top++;
    }
    for(i=BynaryNumber.top; i>=0; i--)
        cout<<BynaryNumber.a[i];
}
```

# 3 NGĂN XẾP - STACK

## Cài đặt ngăn xếp bằng danh sách liên kết đơn

Theo tính chất của DSLK đơn, việc bổ sung và loại bỏ 1 phần tử thực hiện đơn giản và nhanh nhất khi phần tử đó nằm đầu danh sách. Do vậy, ta sẽ chọn cách lưu trữ của ngăn xếp theo thứ tự như sau:

- Phần tử đầu danh sách là đỉnh của stack.
- Phần tử cuối danh sách là đáy của stack.
- Để bổ sung 1 phần tử mới vào stack ta thêm nó vào đầu danh sách.
- Để lấy 1 phần tử ra khỏi stack ta lấy giá trị nút đầu tiên và loại nó ra khỏi danh sách.



# 3 NGĂN XẾP - STACK

**Một số ứng dụng của ngăn xếp:**

- Đảo ngược xâu kí tự.
- Tính giá trị biểu thức dạng hậu tố (postfix).
- Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix).
- Khử đệ quy lui.
- ....

# 3 NGĂN XẾP - STACK

**Đảo ngược xâu kí tự:**

Bài toán đảo ngược xâu kí tự yêu cầu hiển thị các kí tự của xâu theo chiều ngược lại (ví dụ: Xâu ngược của xâu “lap trinh” là “hnirt pal”).

Để giải quyết bài toán ta chỉ việc duyệt từ đầu đến cuối xâu và đưa vào ngăn xếp. Sau đó ta chỉ việc lấy các kí tự ra khỏi ngăn xếp và hiển thị chúng lên màn hình.

# 3 NGĂN XẾP - STACK

**Biểu thức dạng hậu tố (postfix):**

Biểu thức toán học mà con người vẫn hay dùng gọi là biểu thức dạng trung tố (infix). Ví dụ biểu thức:

| ( | 4 | * | ( | ( | ( | 6 | - | 3 | ) | * | ( | 8 | - | 3 | ) | ) | + | 1 | ) | ) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Tuy nhiên đối với biểu thức dạng này thì việc tính toán đối với máy tính rất khó khăn do vậy để dễ dàng hơn cho máy tính trong việc tính toán người ta đưa ra dạng trình bày biểu thức toán học khác gọi là biểu thức dạng hậu tố (postfix). Theo cách trình bày này toán tử không nằm giữa 2 toán hạng mà nằm ngay sau 2 toán hạng. Chẳng hạn biểu thức trên có thể viết dưới dạng hậu tố như sau:

| 4 | 6 | 3 | - | 8 | 3 | - | * | 1 | + | * |
|---|---|---|---|---|---|---|---|---|---|---|

# 3 NGĂN XẾP - STACK

## Biểu thức dạng hậu tố (postfix):

Thuật toán tính giá trị của biểu thức dạng hậu tố như sau: Duyệt biểu thức từ trái qua phải:

- Nếu gặp toán hạng, đưa vào ngăn xếp.
- Nếu gặp toán tử, lấy ra 2 toán hạng từ ngăn xếp và sử dụng toán tử trên để tính, đưa kết quả vào ngăn xếp.

# 3 NGĂN XẾP - STACK

**Biểu thức dạng hậu tố (postfix):**

**Ví dụ**: Tính giá trị của biểu thức hậu tố:

**Stack**



# 3 NGĂN XẾP - STACK

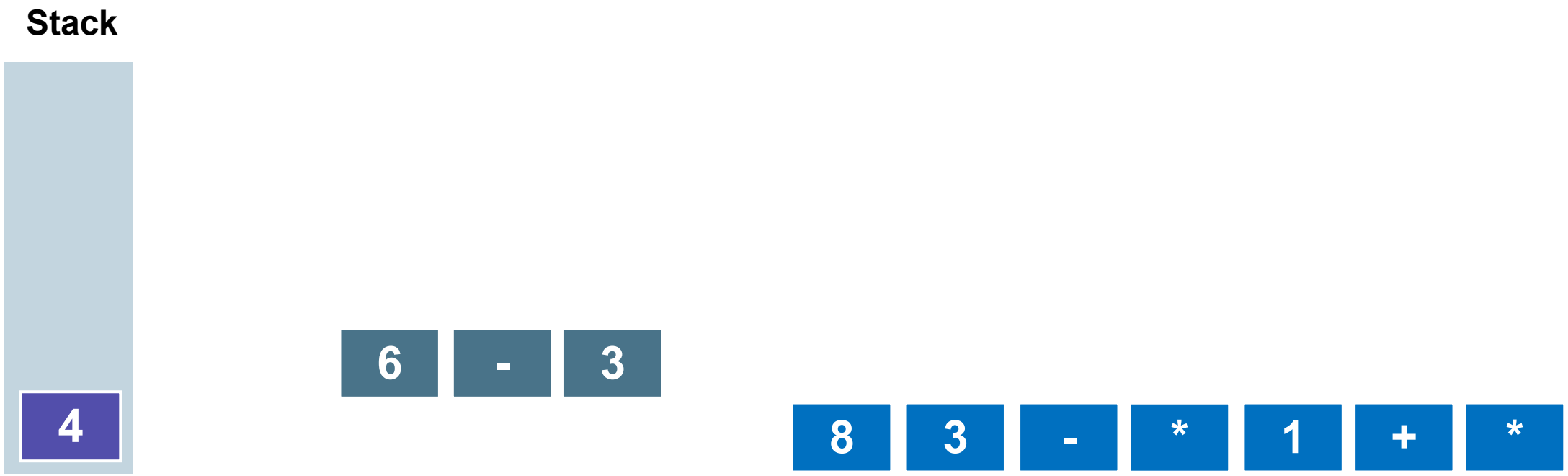
**Biểu thức dạng hậu tố (postfix):**

**Ví dụ**: Tính giá trị của biểu thức hậu tố:

# 3 NGĂN XẾP - STACK

**Biểu thức dạng hậu tố (postfix):**

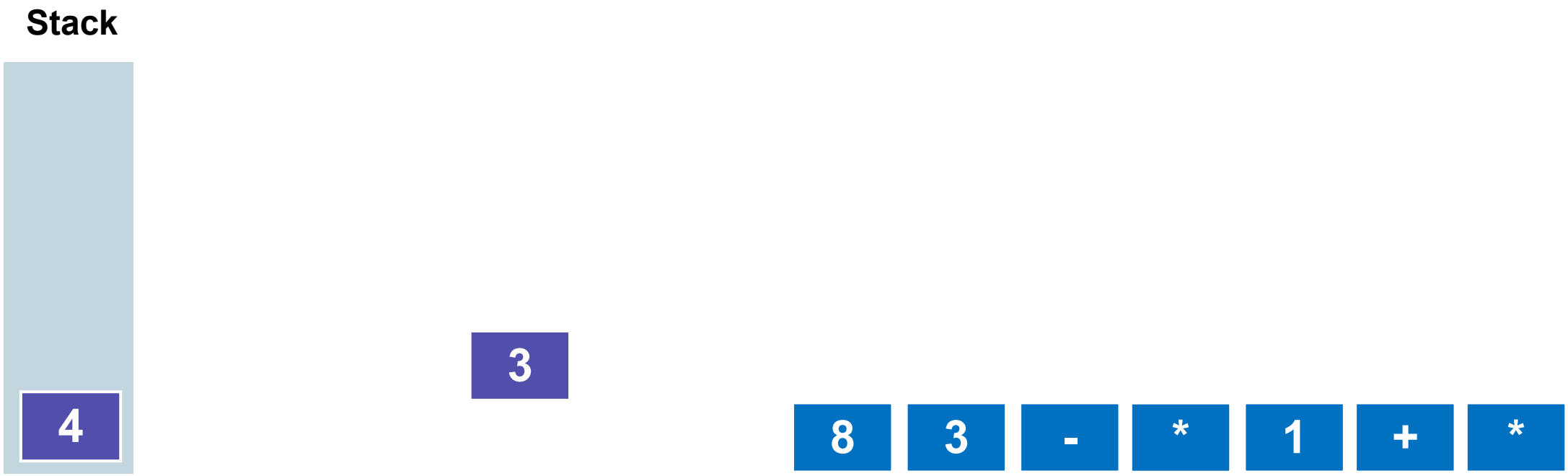
**Ví dụ**: Tính giá trị của biểu thức hậu tố:



# 3 NGĂN XẾP - STACK

**Biểu thức dạng hậu tố (postfix):**

**Ví dụ**: Tính giá trị của biểu thức hậu tố:



# 3 NGĂN XẾP - STACK

**Biểu thức dạng hậu tố (postfix):**

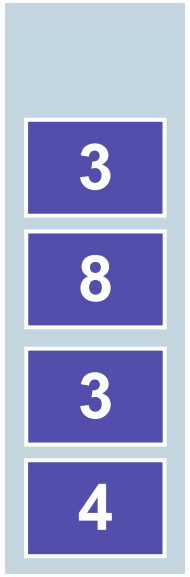
**Ví dụ**: Tính giá trị của biểu thức hậu tố:

# 3 NGĂN XẾP - STACK

**Biểu thức dạng hậu tố (postfix):**

**Ví dụ**: Tính giá trị của biểu thức hậu tố:

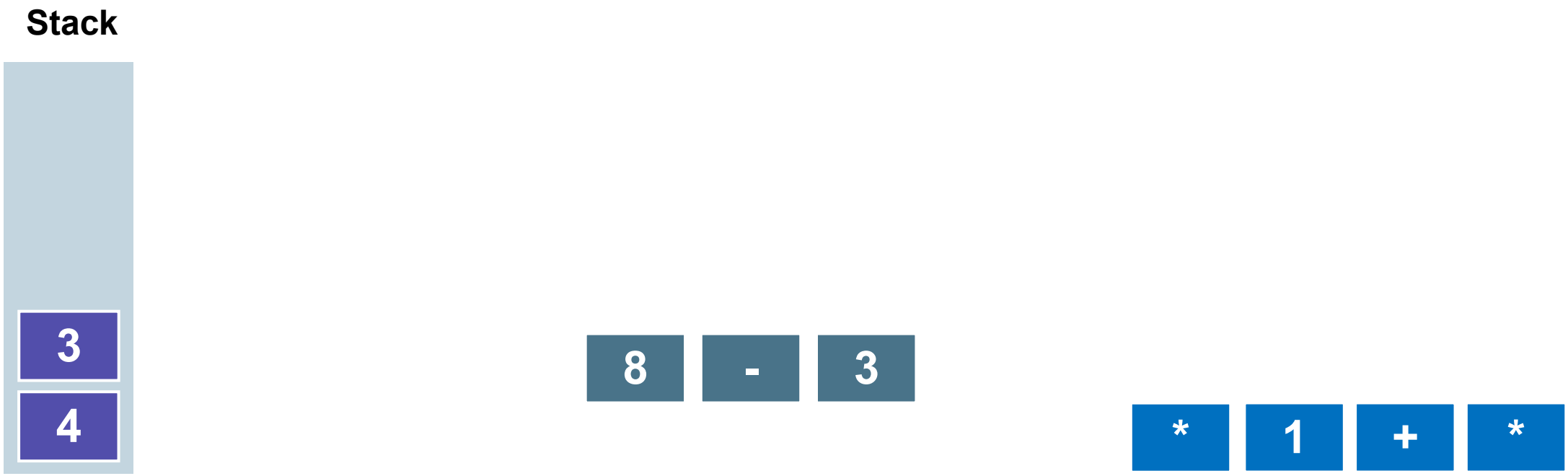
**Stack**




# 3 NGĂN XẾP - STACK

**Biểu thức dạng hậu tố (postfix):**

**Ví dụ**: Tính giá trị của biểu thức hậu tố:



# 3 NGĂN XẾP - STACK

**Biểu thức dạng hậu tố (postfix):**

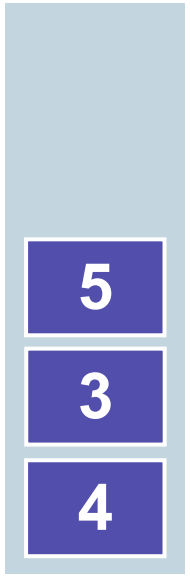
**Ví dụ**: Tính giá trị của biểu thức hậu tố:

# 3 NGĂN XẾP - STACK

**Biểu thức dạng hậu tố (postfix):**

**Ví dụ**: Tính giá trị của biểu thức hậu tố:

**Stack**




# 3 NGĂN XẾP - STACK

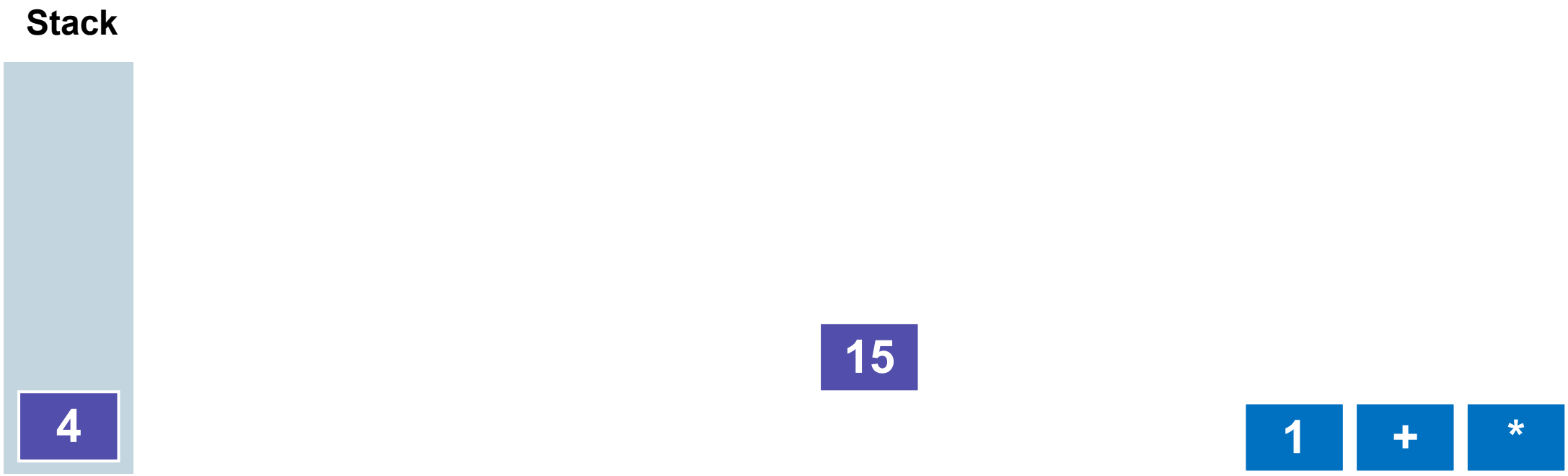
**Biểu thức dạng hậu tố (postfix):**

**Ví dụ**: Tính giá trị của biểu thức hậu tố:

# 3 NGĂN XẾP - STACK

**Biểu thức dạng hậu tố (postfix):**

**Ví dụ**: Tính giá trị của biểu thức hậu tố:



# 3 NGĂN XẾP - STACK

**Biểu thức dạng hậu tố (postfix):**

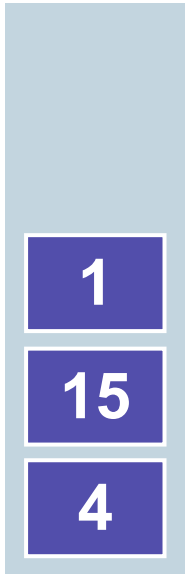
**Ví dụ**: Tính giá trị của biểu thức hậu tố:

# 3 NGĂN XẾP - STACK

**Biểu thức dạng hậu tố (postfix):**

**Ví dụ**: Tính giá trị của biểu thức hậu tố:

**Stack**




# 3 NGĂN XẾP - STACK

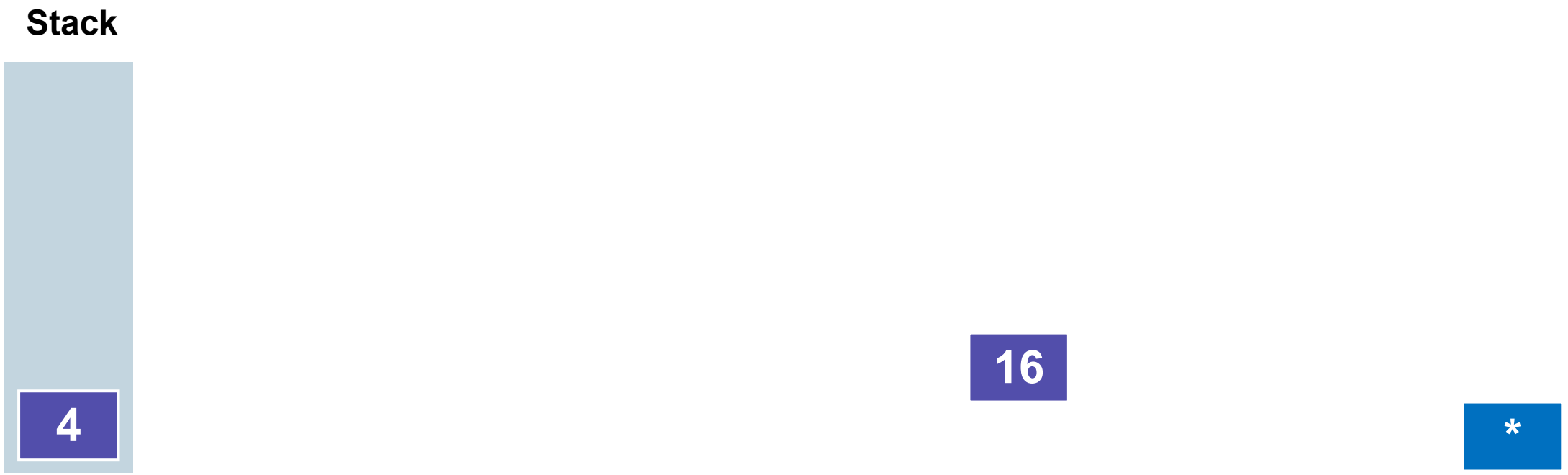
**Biểu thức dạng hậu tố (postfix):**

**Ví dụ**: Tính giá trị của biểu thức hậu tố:

# 3 NGĂN XẾP - STACK

**Biểu thức dạng hậu tố (postfix):**

**Ví dụ**: Tính giá trị của biểu thức hậu tố:



# 3 NGĂN XẾP - STACK

**Biểu thức dạng hậu tố (postfix):**

**Ví dụ**: Tính giá trị của biểu thức hậu tố:

# 3 NGĂN XẾP - STACK

**Biểu thức dạng hậu tố (postfix):**

**Ví dụ**: Tính giá trị của biểu thức hậu tố:

**Stack**



4 * 16

# 3 NGĂN XẾP - STACK

**Biểu thức dạng hậu tố (postfix):**

**Ví dụ**: Tính giá trị của biểu thức hậu tố:

**Stack**



64

# 3 NGĂN XẾP - STACK

**Biểu thức dạng hậu tố (postfix):**

**Ví dụ**: Tính giá trị của biểu thức hậu tố:

| 4 | 6 | 3 | - | 8 | 3 | - | * | 1 | + | * | = | 64 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

Để chuyển đổi 1 biểu thức từ dạng trung tố sang dạng hậu tố ta duyệt biểu thức từ trái qua phải:

- Nếu gặp dấu mở ngoặc **(** thì bỏ qua.
- Nếu gặp toán hạng, đưa vào biểu thức mới.
- Nếu gặp toán tử, đưa vào ngăn xếp.
- Nếu gặp dấu đóng ngoặc **)** , lấy toán tử ra khỏi ngăn xếp và đưa vào biểu thức mới.

# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

**Ví dụ**: Chuyển đổi biểu thức sau từ trung tố sang hậu tố

| ( | 4 | * | ( | ( | ( | 6 | - | 3 | ) | * | ( | 8 | - | 3 | ) | ) | + | 1 | ) | ) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

**Ví dụ**: Chuyển đổi biểu thức sau từ trung tố sang hậu tố

| ( | 4 | * | ( | ( | ( | 6 | - | 3 | ) | * | ( | 8 | - | 3 | ) | ) | + | 1 | ) | ) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

**Stack** **Biểu thức hậu tố**

# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

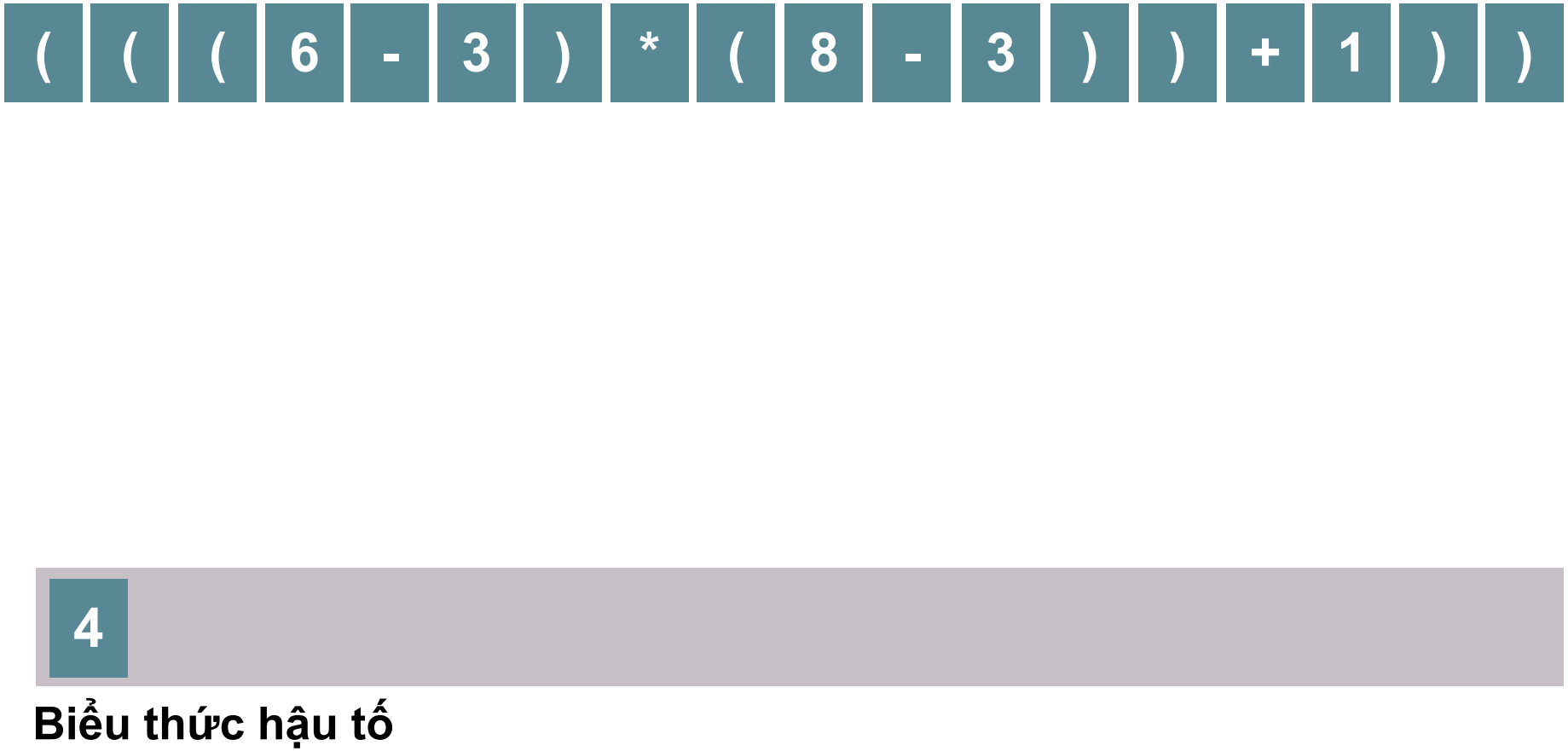
**Ví dụ**: Chuyển đổi biểu thức sau từ trung tố sang hậu tố

| * | ( | ( | ( | 6 | - | 3 | ) | * | ( | 8 | - | 3 | ) | ) | + | 1 | ) | ) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

**Ví dụ**: Chuyển đổi biểu thức sau từ trung tố sang hậu tố

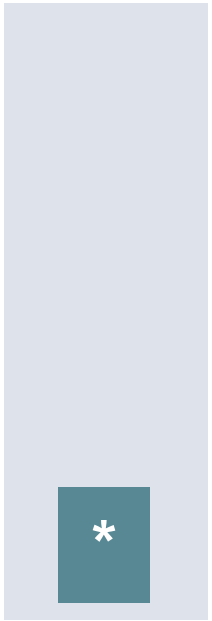


# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

**Ví dụ**: Chuyển đổi biểu thức sau từ trung tố sang hậu tố

| 6 | - | 3 | ) | * | ( | 8 | - | 3 | ) | ) | + | 1 | ) | ) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|




**Stack**


**Biểu thức hậu tố**

# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

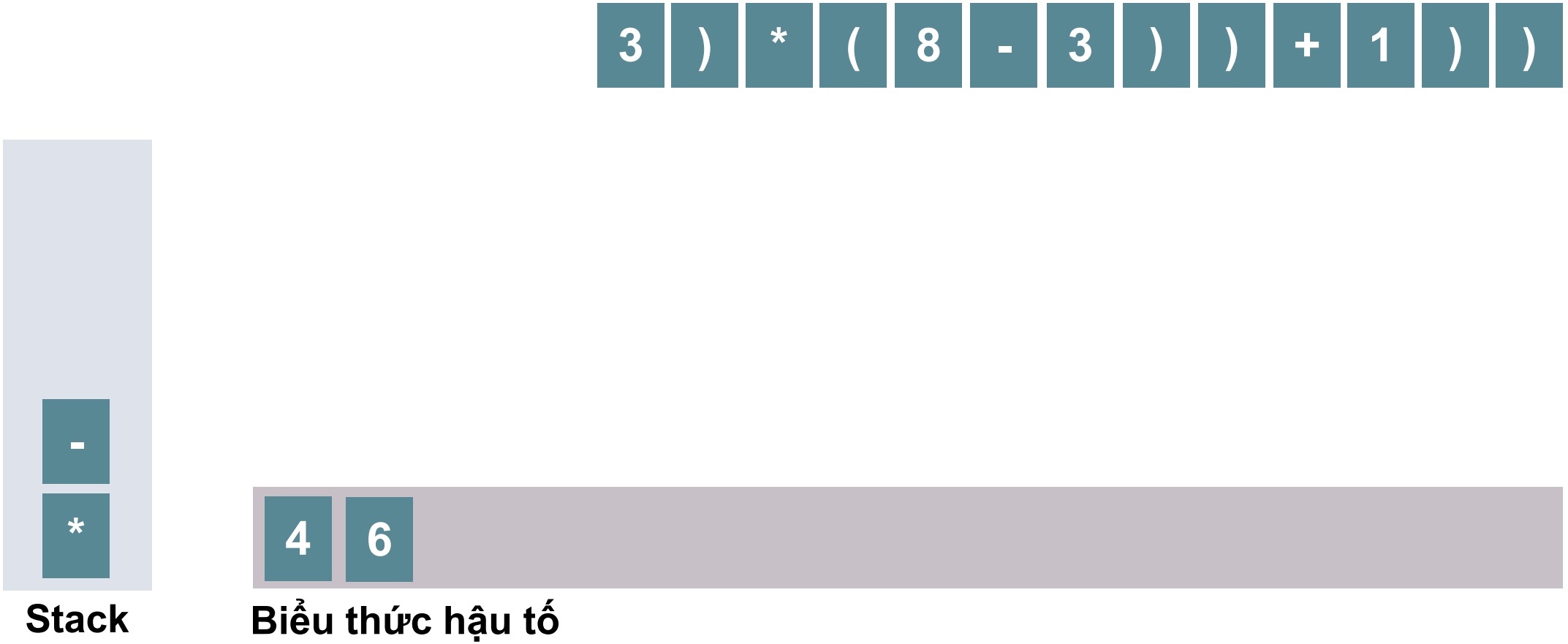
**Ví dụ**: Chuyển đổi biểu thức sau từ trung tố sang hậu tố

| - | 3 | ) | * | ( | 8 | - | 3 | ) | ) | + | 1 | ) | ) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

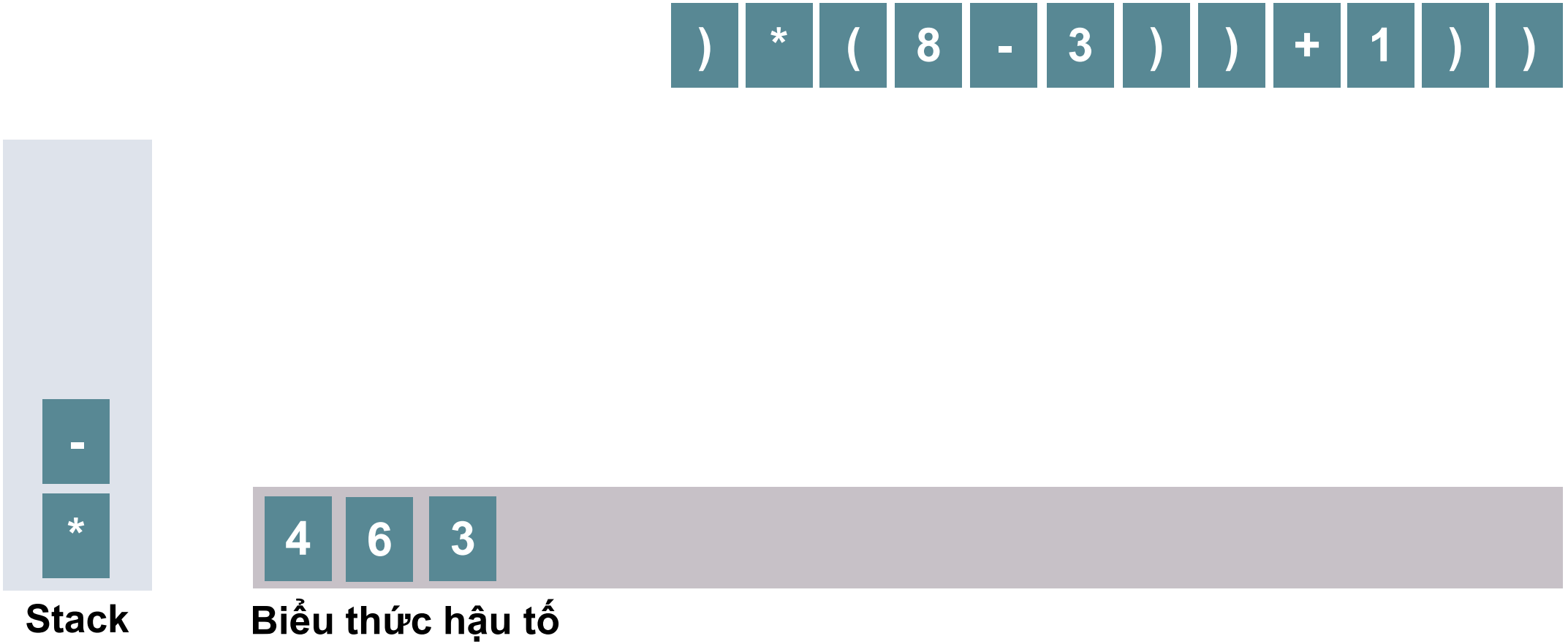
**Ví dụ**: Chuyển đổi biểu thức sau từ trung tố sang hậu tố



# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

**Ví dụ**: Chuyển đổi biểu thức sau từ trung tố sang hậu tố

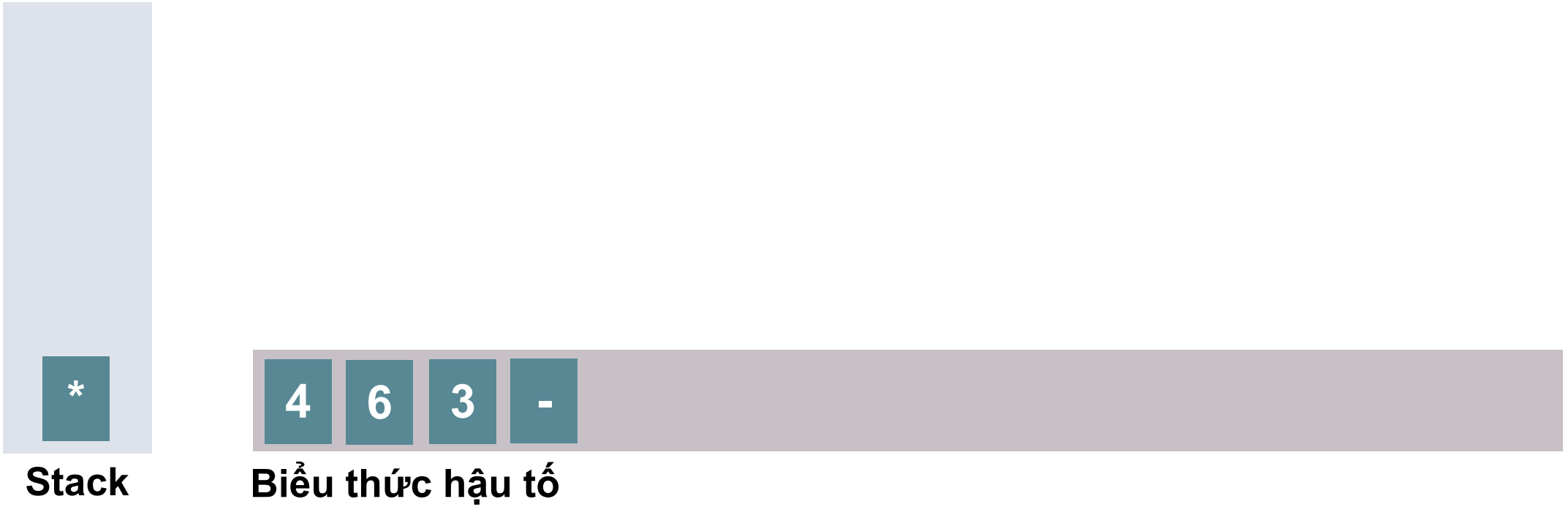


# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

**Ví dụ**: Chuyển đổi biểu thức sau từ trung tố sang hậu tố

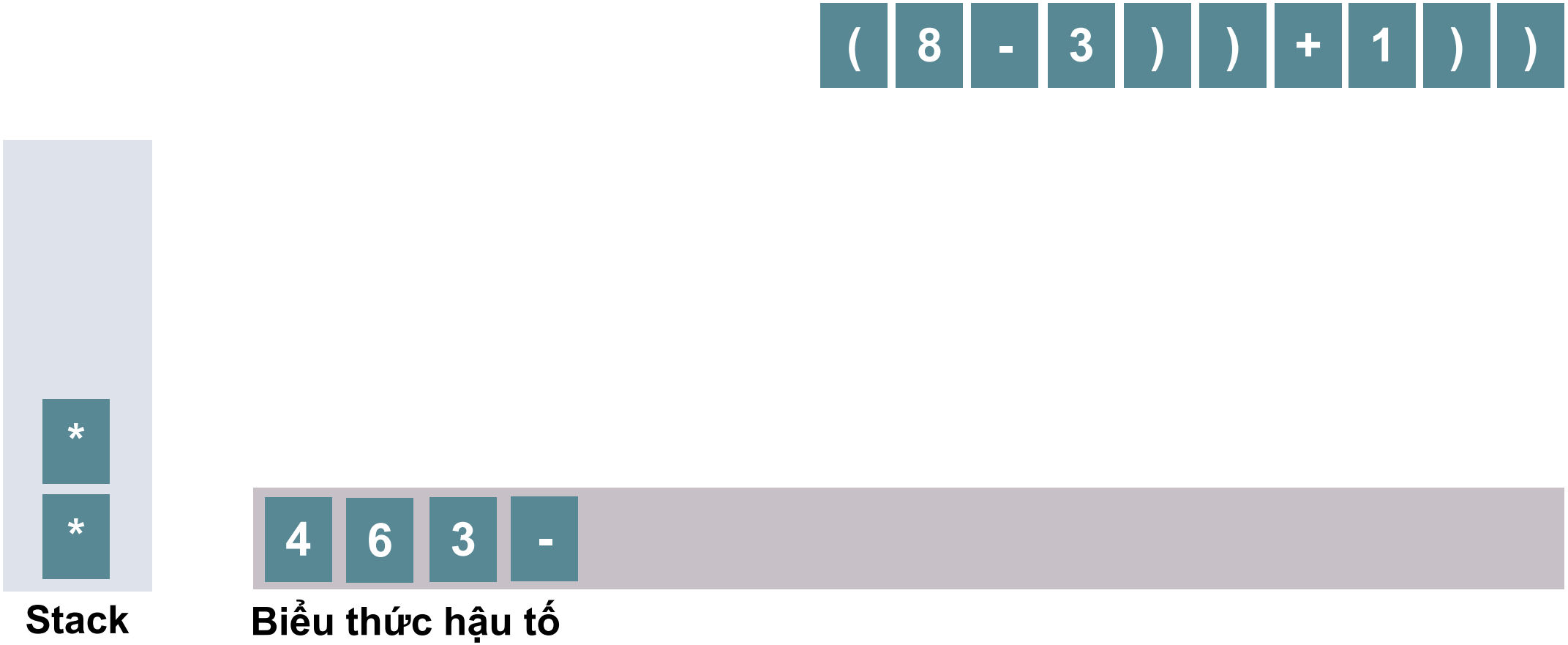
| * | ( | 8 | - | 3 | ) | ) | + | 1 | ) | ) |
|---|---|---|---|---|---|---|---|---|---|---|



# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

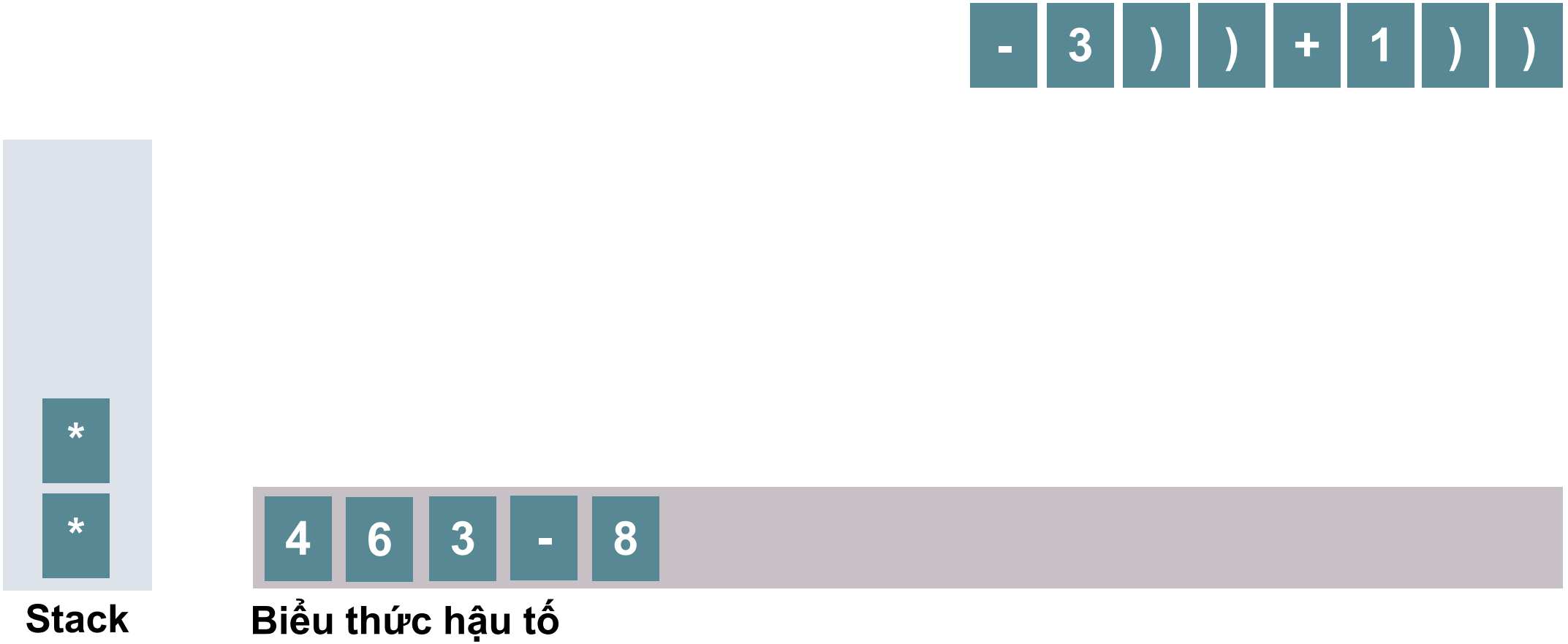
**Ví dụ**: Chuyển đổi biểu thức sau từ trung tố sang hậu tố



# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

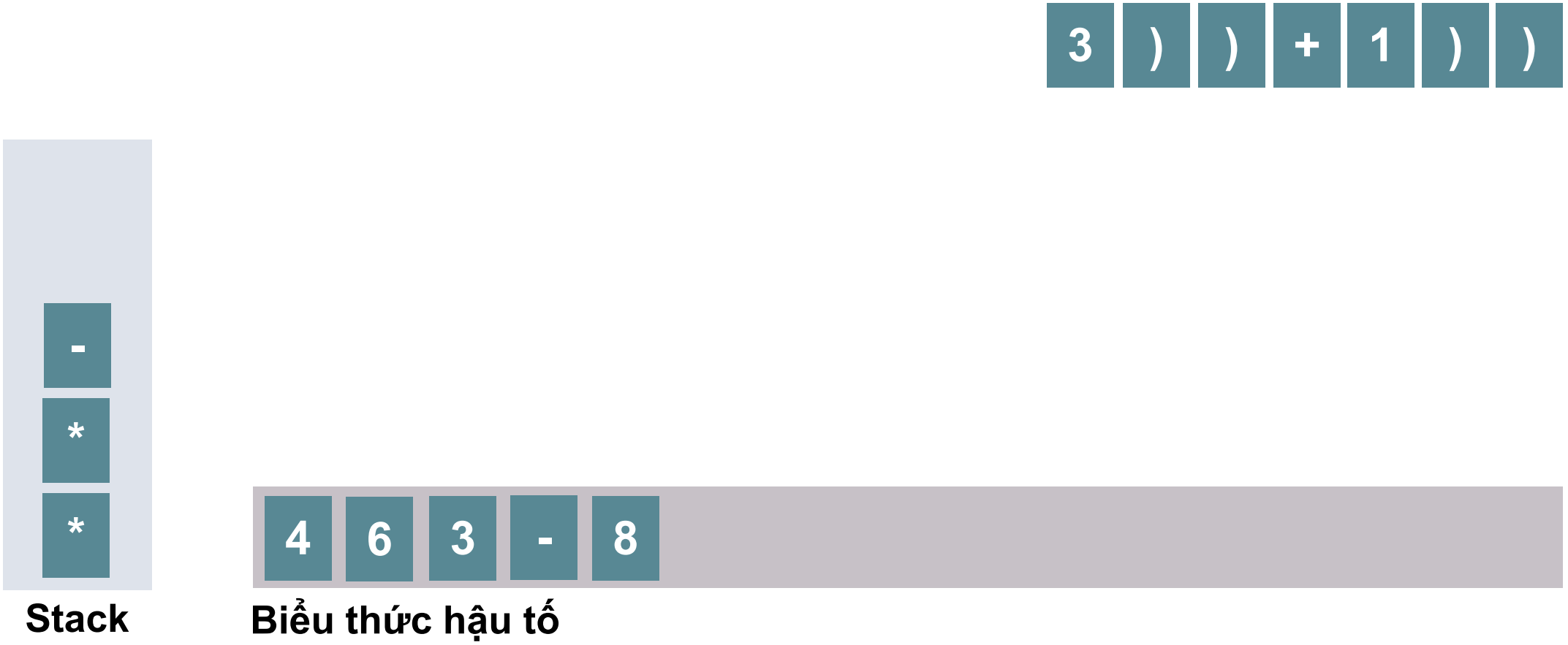
**Ví dụ**: Chuyển đổi biểu thức sau từ trung tố sang hậu tố



# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

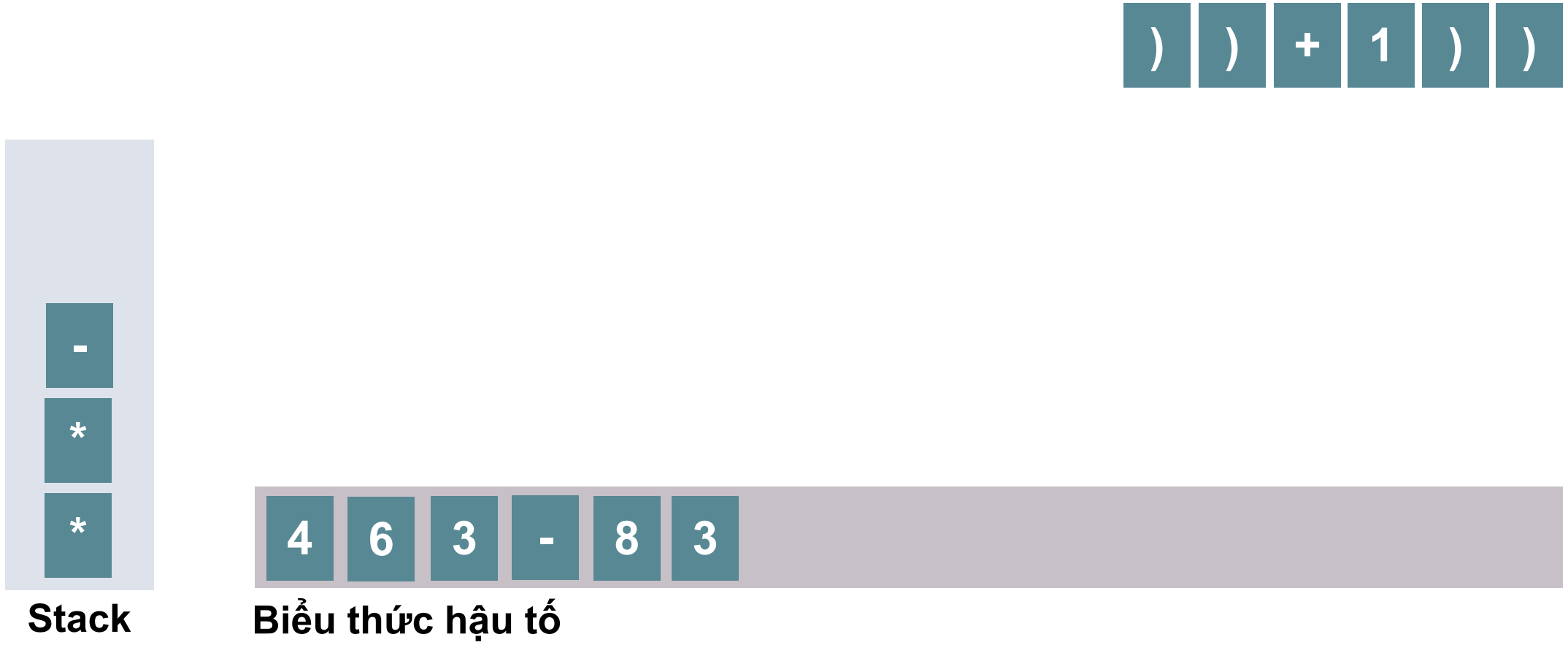
**Ví dụ**: Chuyển đổi biểu thức sau từ trung tố sang hậu tố



# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

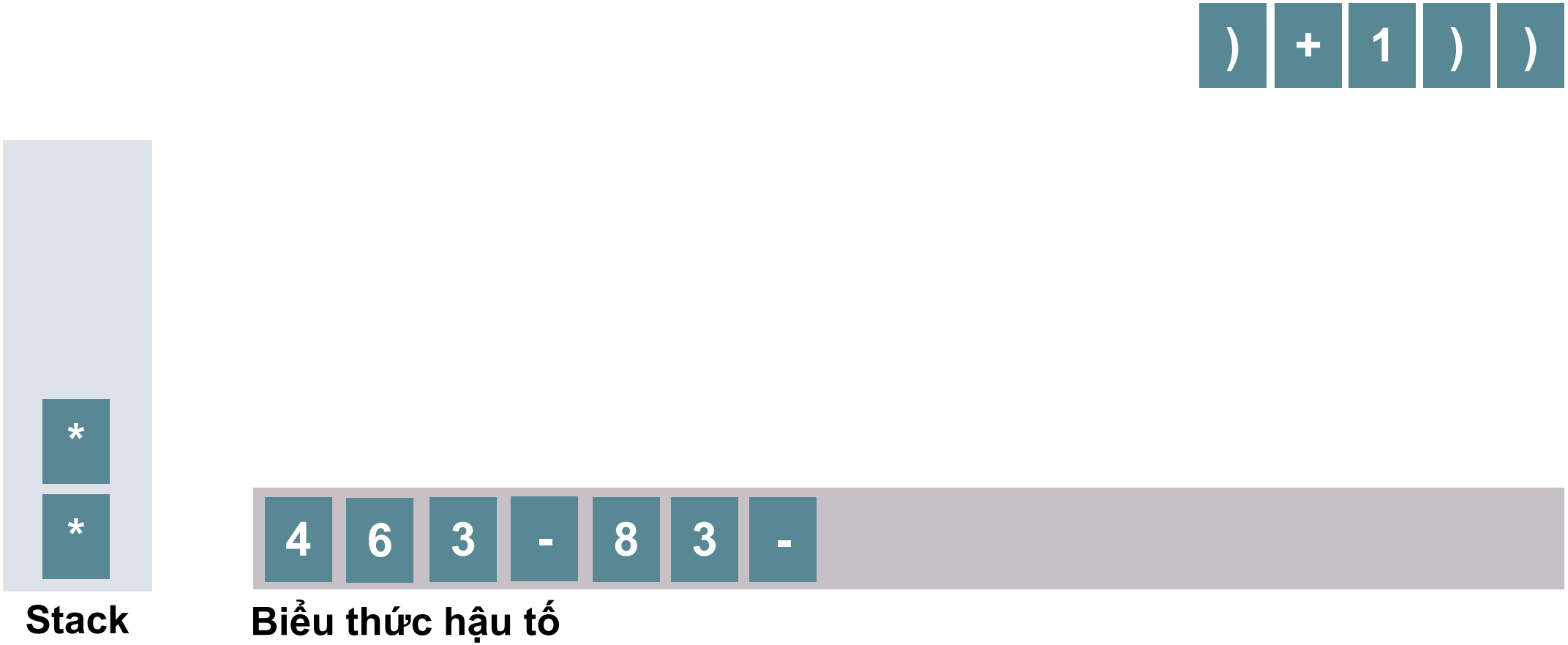
**Ví dụ**: Chuyển đổi biểu thức sau từ trung tố sang hậu tố



# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

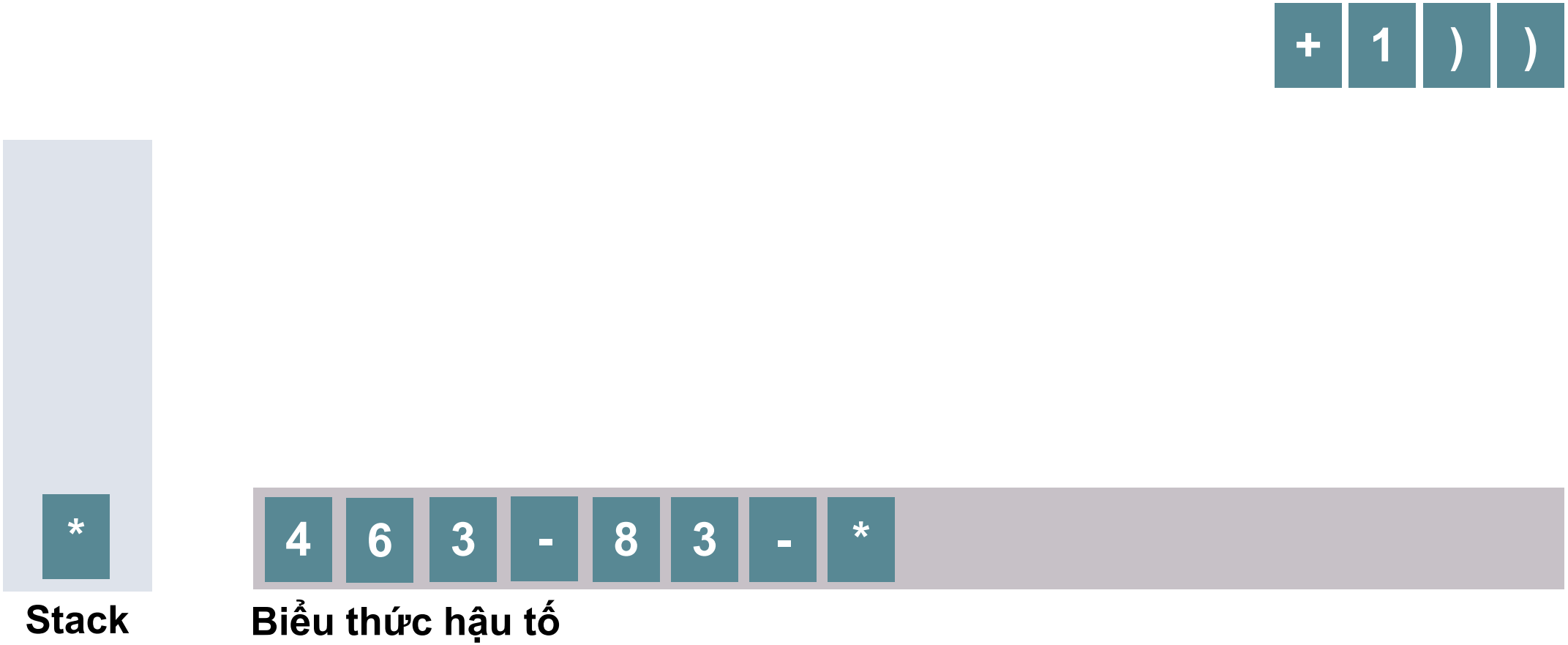
**Ví dụ**: Chuyển đổi biểu thức sau từ trung tố sang hậu tố



# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

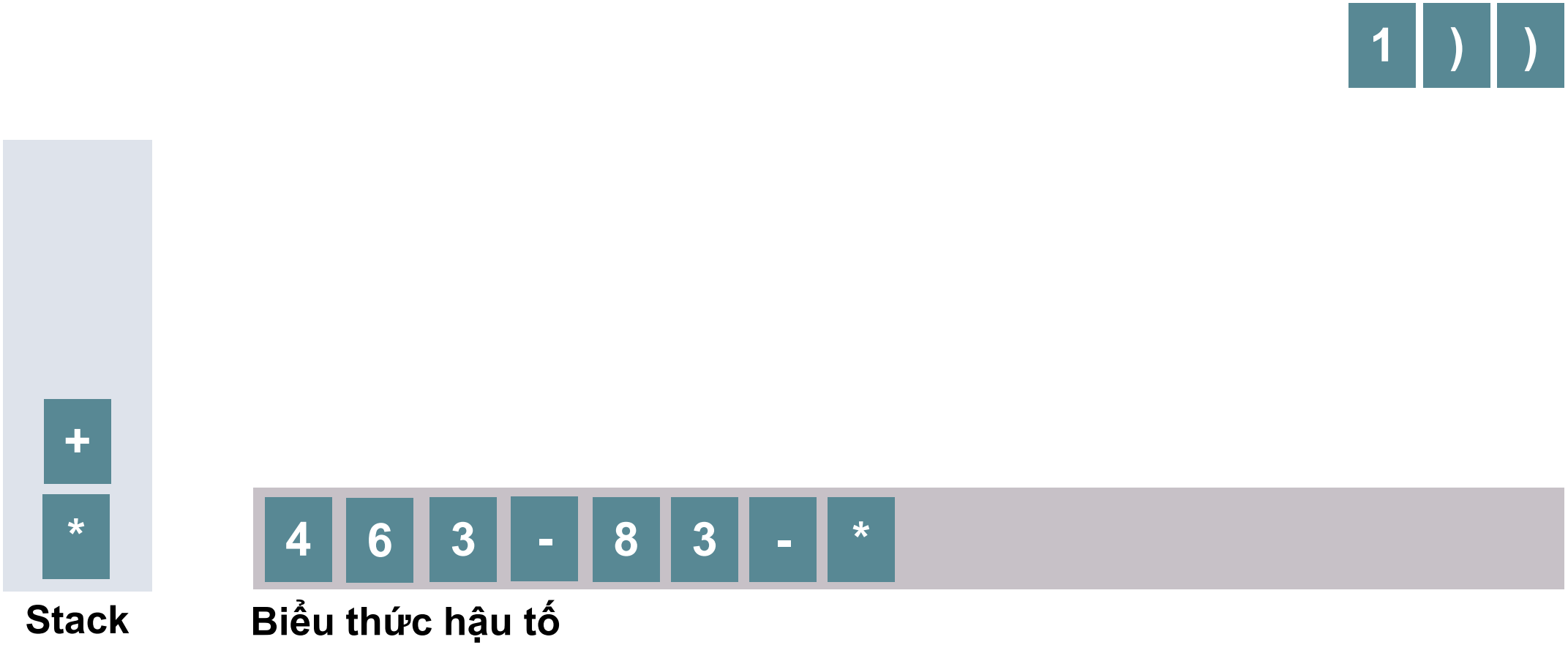
**Ví dụ**: Chuyển đổi biểu thức sau từ trung tố sang hậu tố



# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

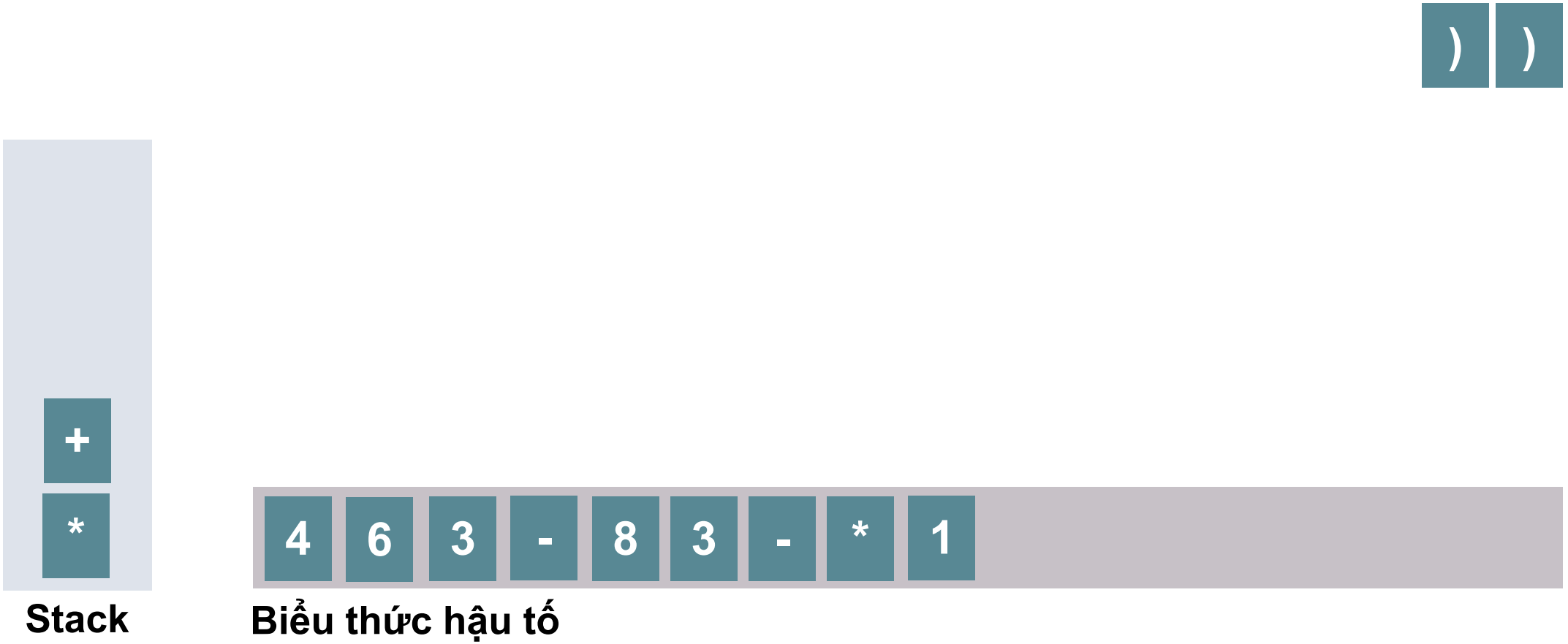
**Ví dụ**: Chuyển đổi biểu thức sau từ trung tố sang hậu tố



# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

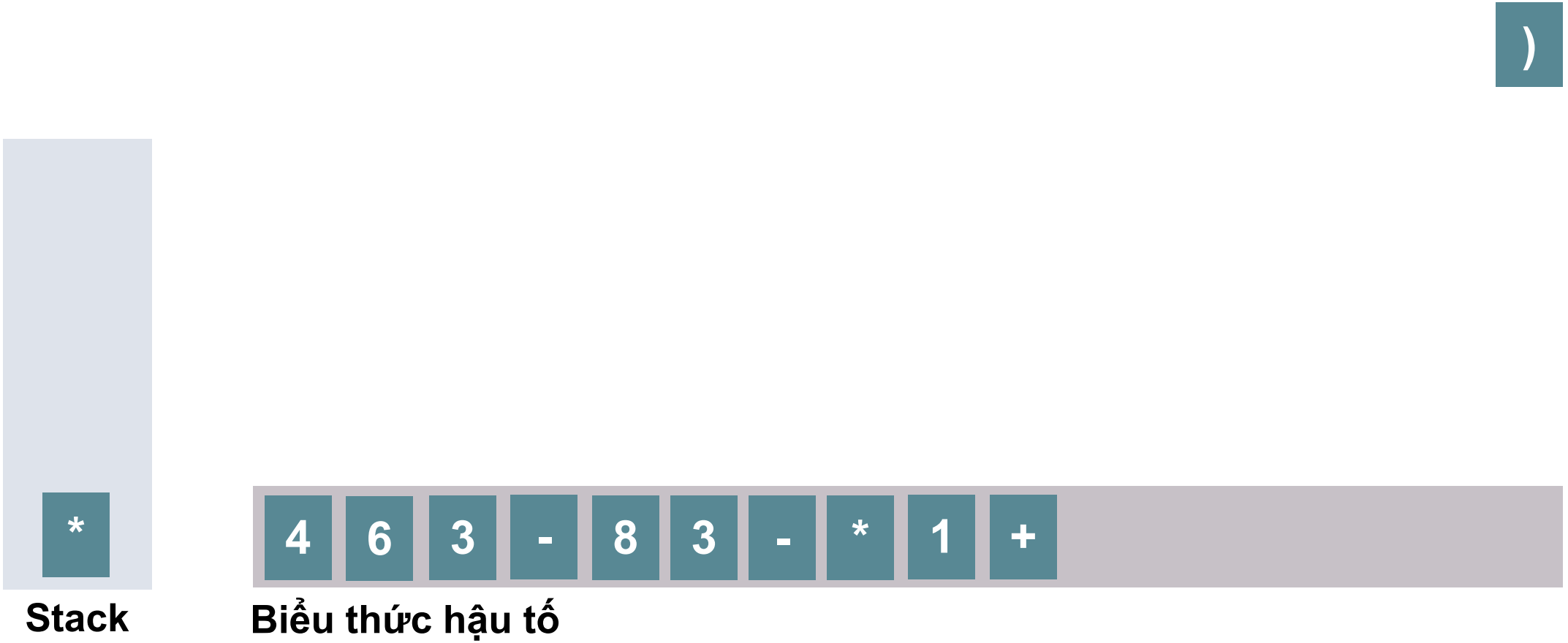
**Ví dụ**: Chuyển đổi biểu thức sau từ trung tố sang hậu tố



# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

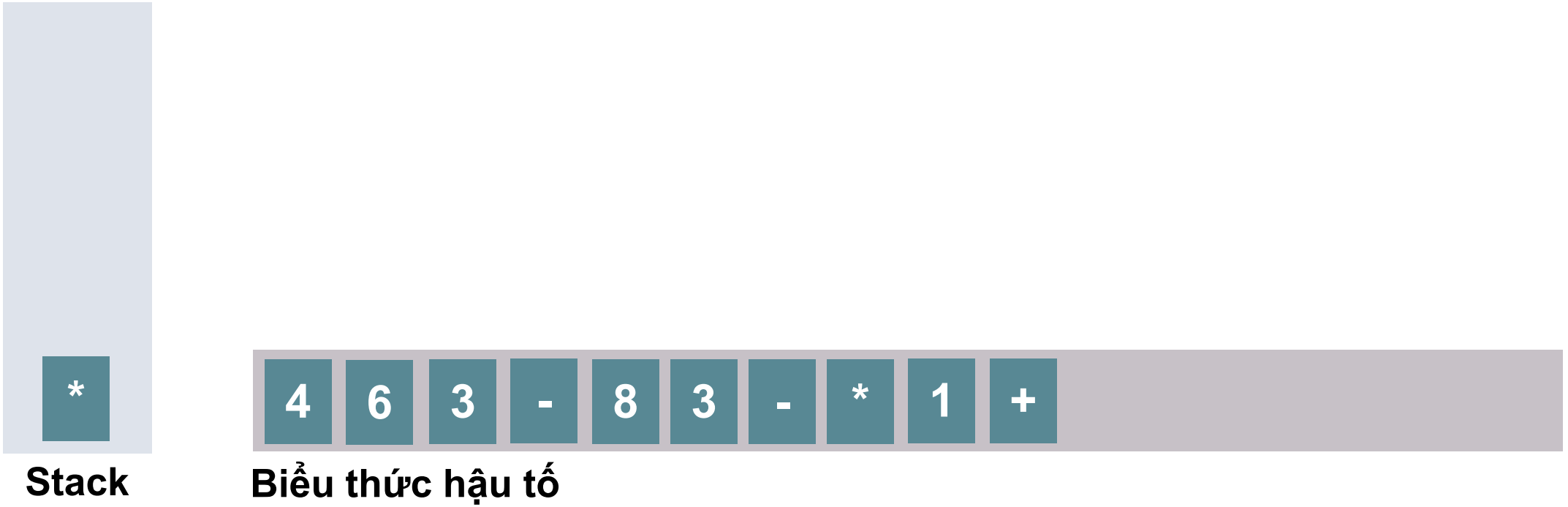
**Ví dụ**: Chuyển đổi biểu thức sau từ trung tố sang hậu tố



# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

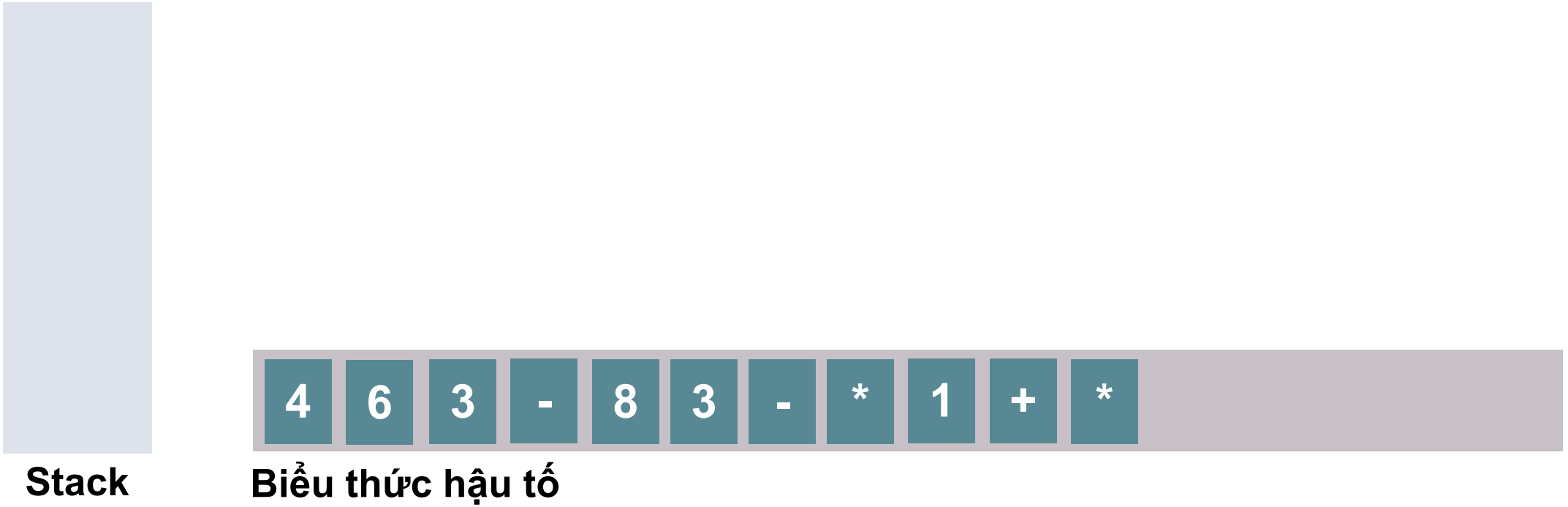
**Ví dụ**: Chuyển đổi biểu thức sau từ trung tố sang hậu tố



# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

**Ví dụ**: Chuyển đổi biểu thức sau từ trung tố sang hậu tố



# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

**Ví dụ**: Chuyển đổi biểu thức sau từ trung tố sang hậu tố


**Biểu thức hậu tố**

# 3 NGĂN XẾP - STACK

**Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):**

**Ví dụ**: Chuyển đổi biểu thức sau từ trung tố sang hậu tố

**Biểu thức trung tố**

| ( | 4 | * | ( | ( | ( | 6 | - | 3 | ) | * | ( | 8 | - | 3 | ) | ) | + | 1 | ) | ) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

**Biểu thức hậu tố**

| 4 | 6 | 3 | - | 8 | 3 | - | * | 1 | + | * |
|---|---|---|---|---|---|---|---|---|---|---|

# 3 NGĂN XẾP - STACK

**Khử đệ quy:**

Khử đệ quy ở đây là biến một thủ tục đệ quy thành một thủ tục chỉ chứa vòng lặp mà không ảnh hưởng gì đến các yếu tố khác, chứ không phải là thay đổi thuật toán.

Khử đệ quy thực chất là chúng ta phải làm công việc của một trình biên dịch đối với một thủ tục, đó là đặt tất cả các giá trị của các biến cục bộ và địa chỉ của chỉ thị kế tiếp vào ngăn xếp (Stack), quy định các giá trị tham số cho thủ tục và chuyển tới vị trí bắt đầu thủ tục, thực hiện lần lượt từng câu lệnh. Sau khi thủ tục hoàn tất thì nó phải lấy ra khỏi ngăn xếp địa chỉ trả về và các giá trị của các biến cục bộ, khôi phục các biến và chuyển tới địa chỉ trả về.

# 3 NGĂN XẾP - STACK

**Khử đệ quy:**

Các bước khử đệ quy dùng stack gồm:

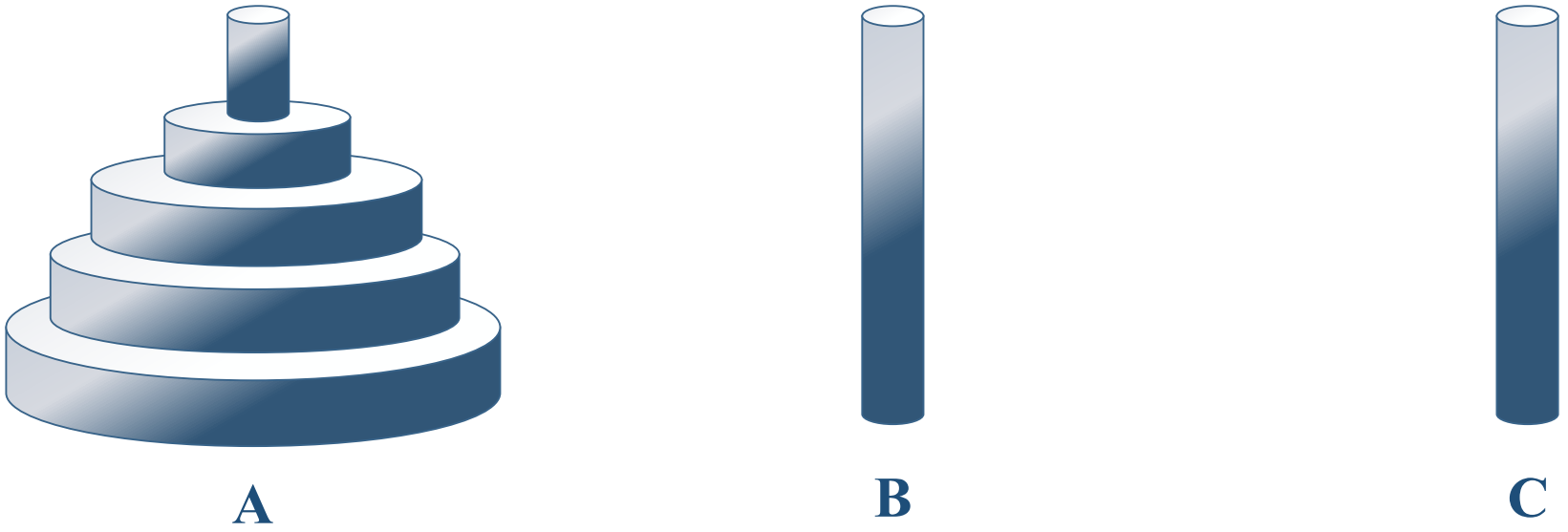
- Bước 1: Lưu các biến cục bộ và địa chỉ trả về.
- Bước 2: Nếu thoả điều kiện ngừng đệ qui thì chuyển sang bước 3. Nếu không thì tính toán từng phần và quay lại bước 1 (đệ qui tiếp).
- Bước 3: Khôi phục lại các biến cục bộ và địa chỉ trả về.

# 3 NGĂN XẾP - STACK

**Khử đệ quy:**

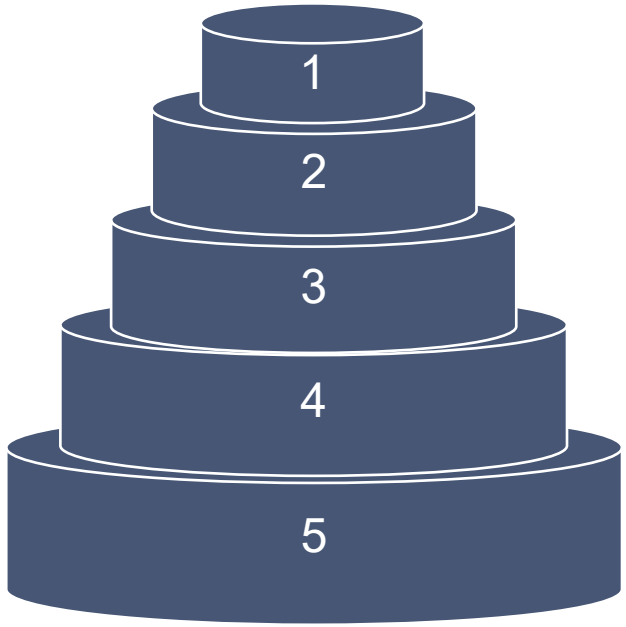
Ví dụ: Bài toán tháp Hà Nội

Bài toàn tháp Hà Nội là trò chơi toán học gồm 3 cọc và n đĩa có kích thước khác nhau. Ban đầu các đĩa được xếp chồng lên nhau trong cọc A. Bài toán đặt ra là phải chuyển toàn bộ chồng đĩa từ A sang B. Mỗi lần thực hiện chuyển một đĩa từ một cọc sang một cọc khác và không được đặt đĩa lớn nằm trên đĩa nhỏ.



# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**

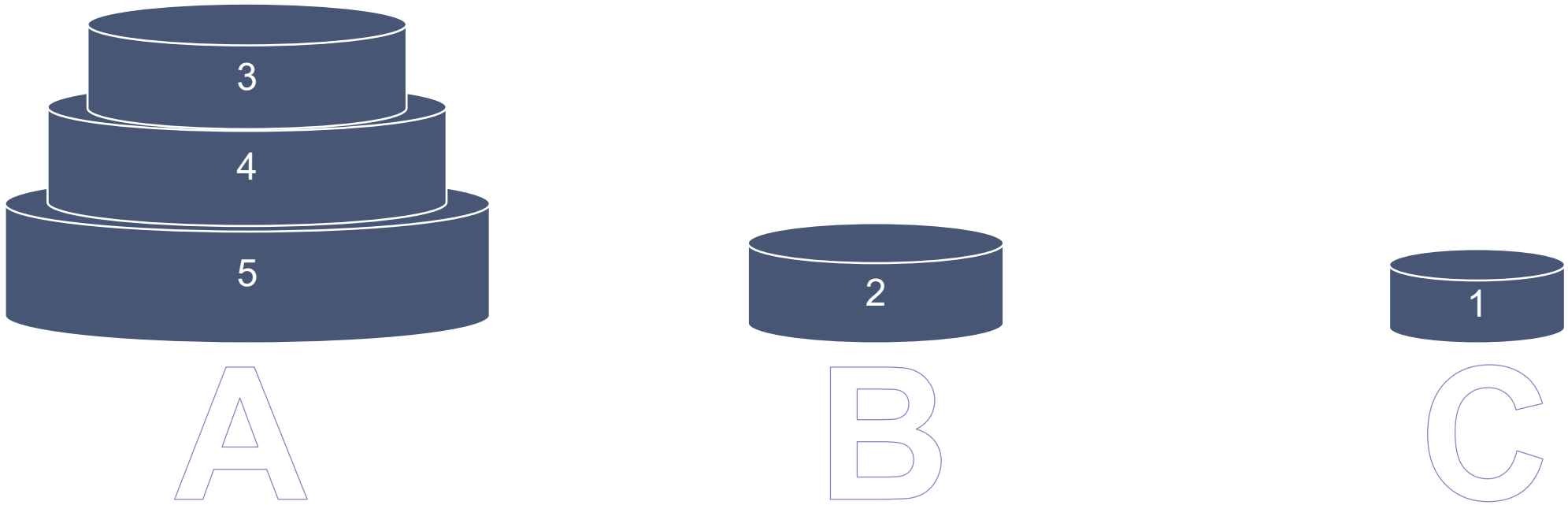


# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**

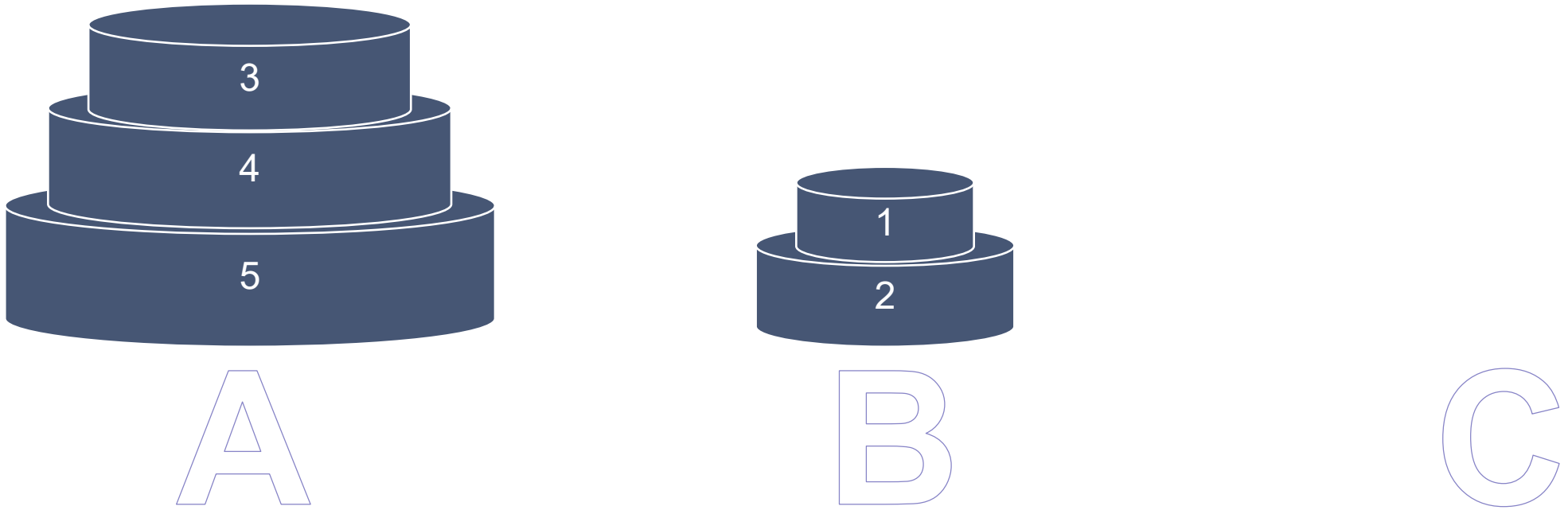
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



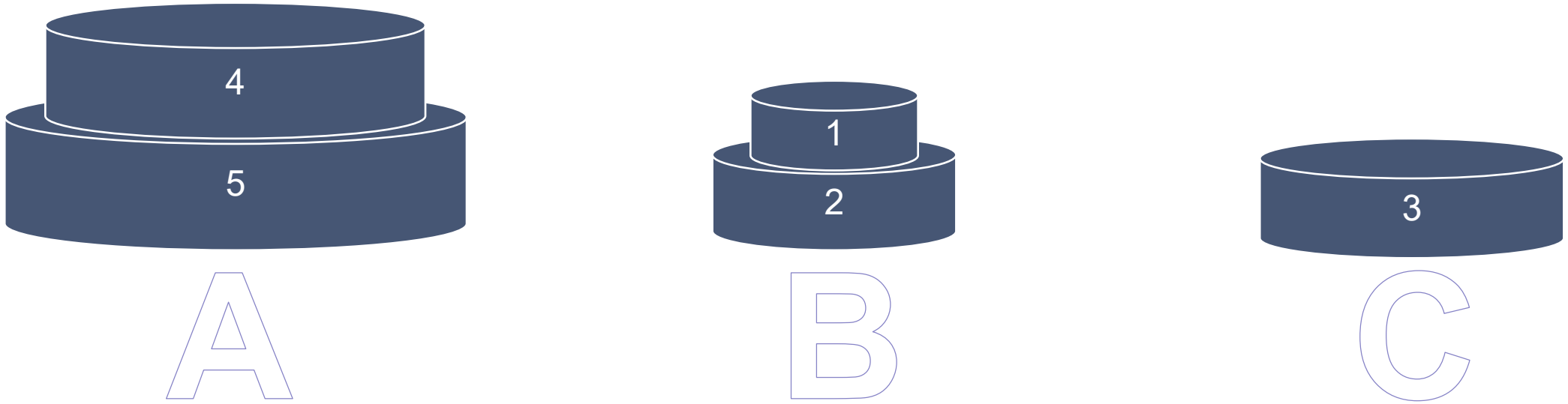
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



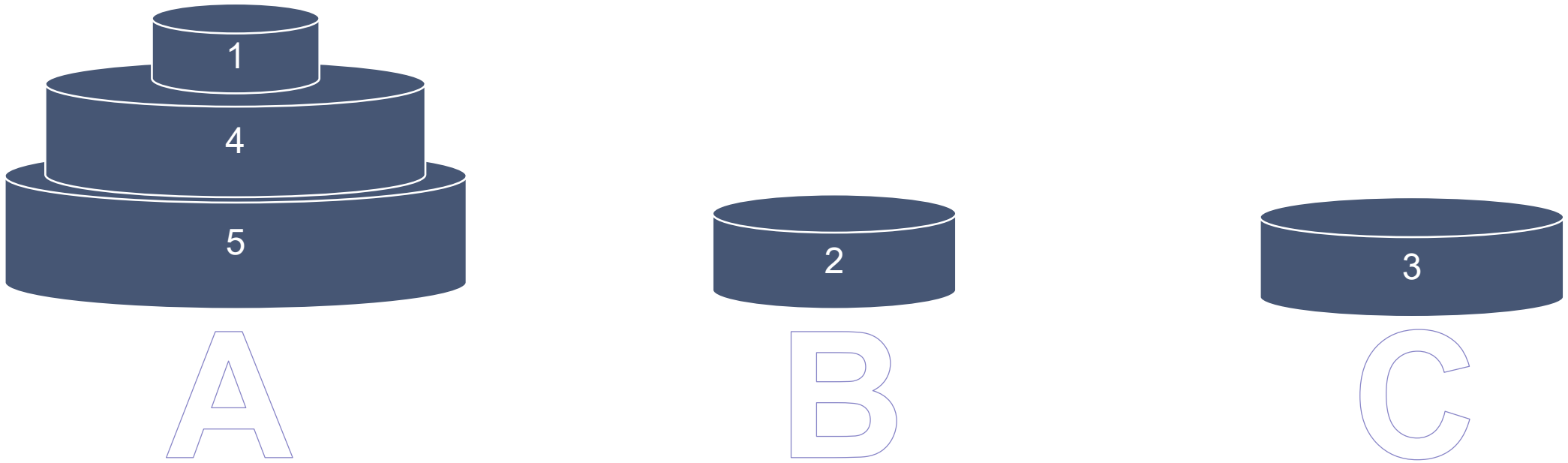
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



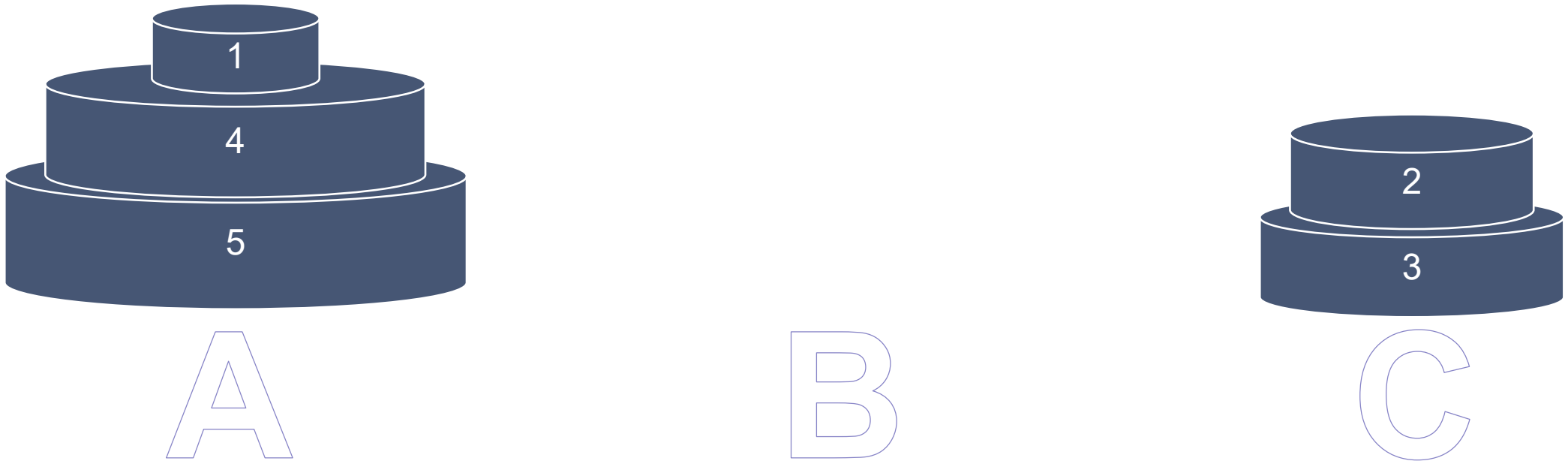
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



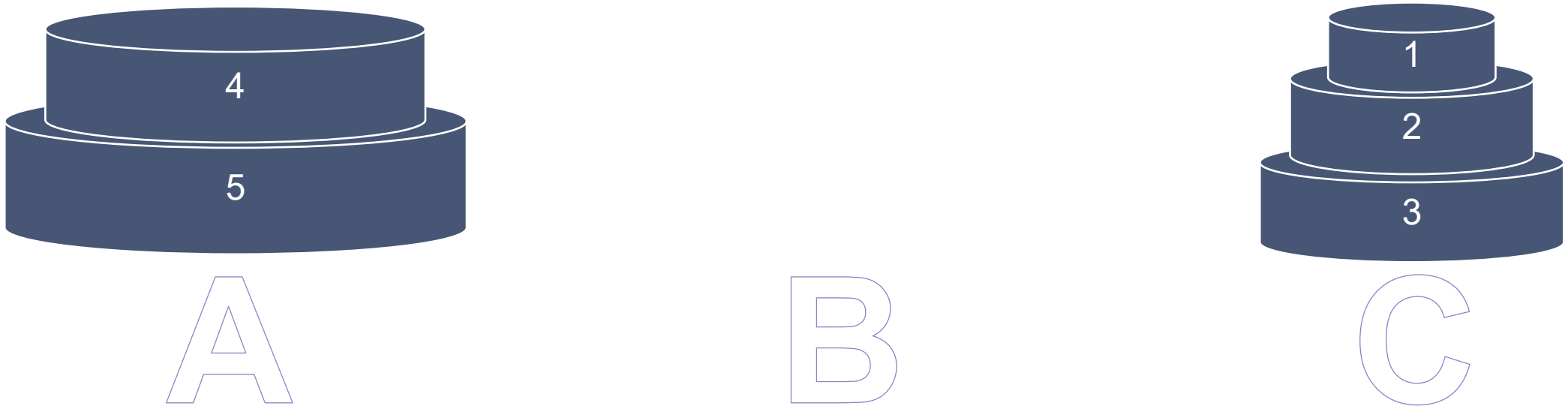
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



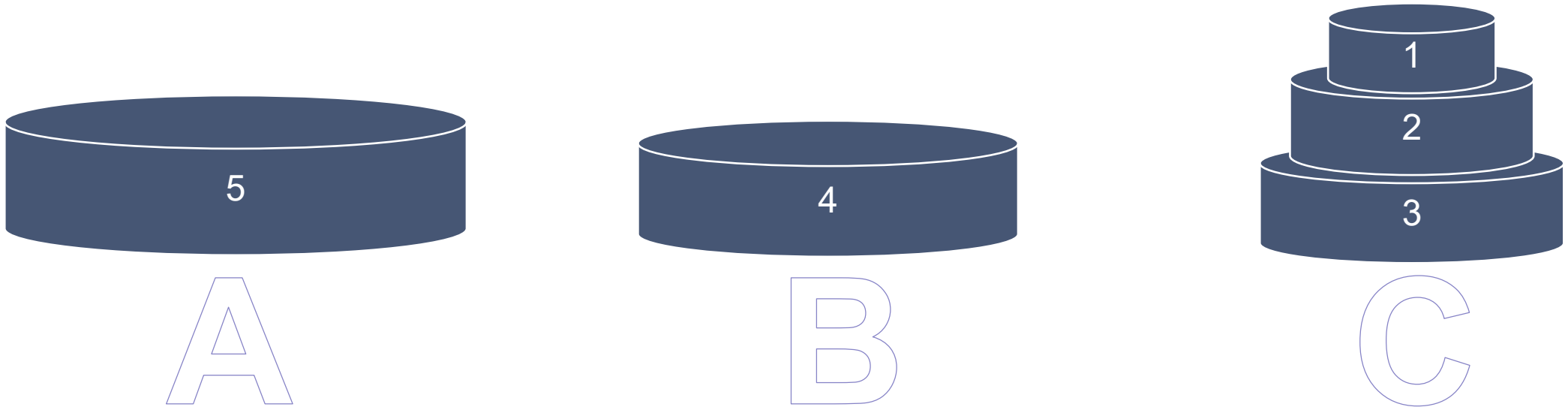
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



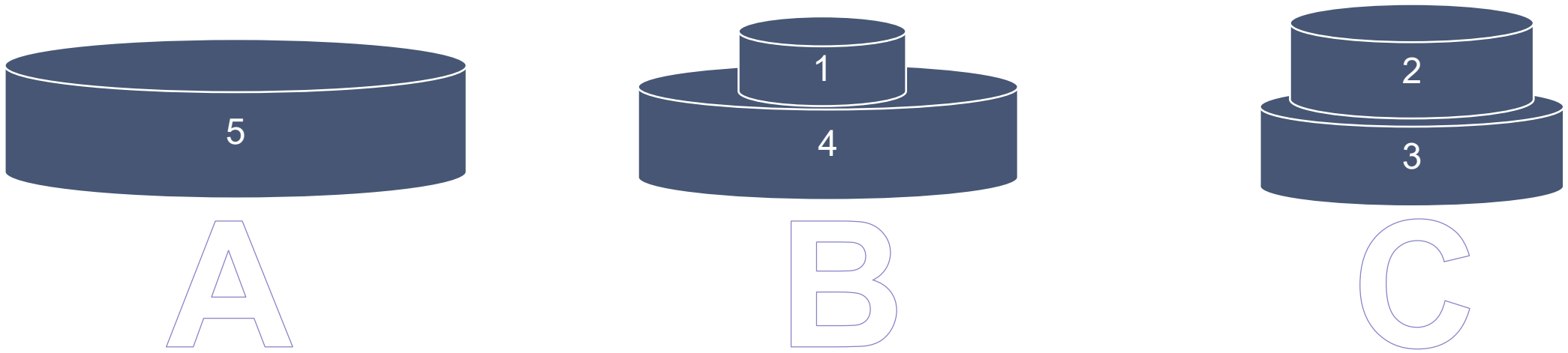
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



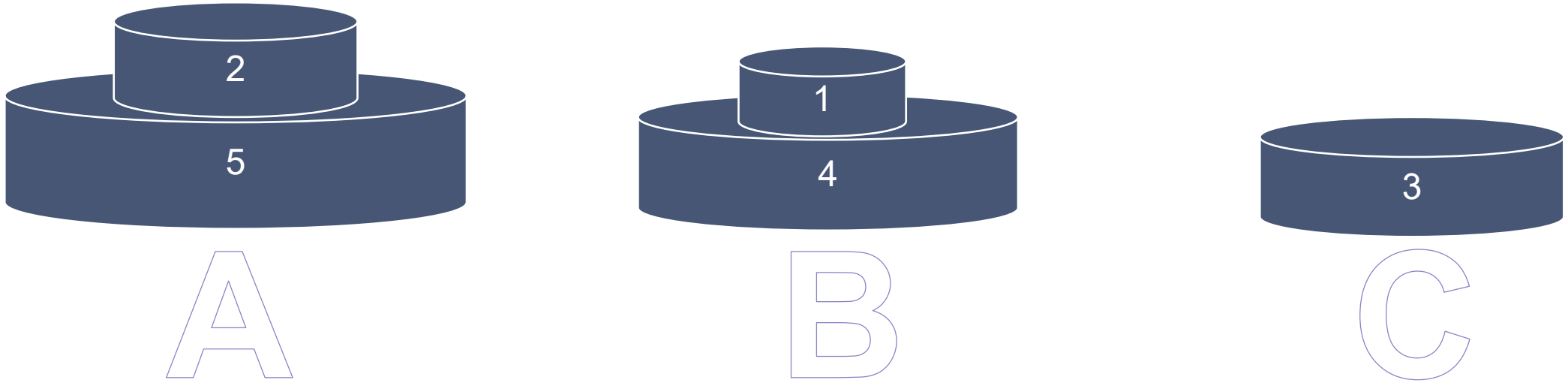
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



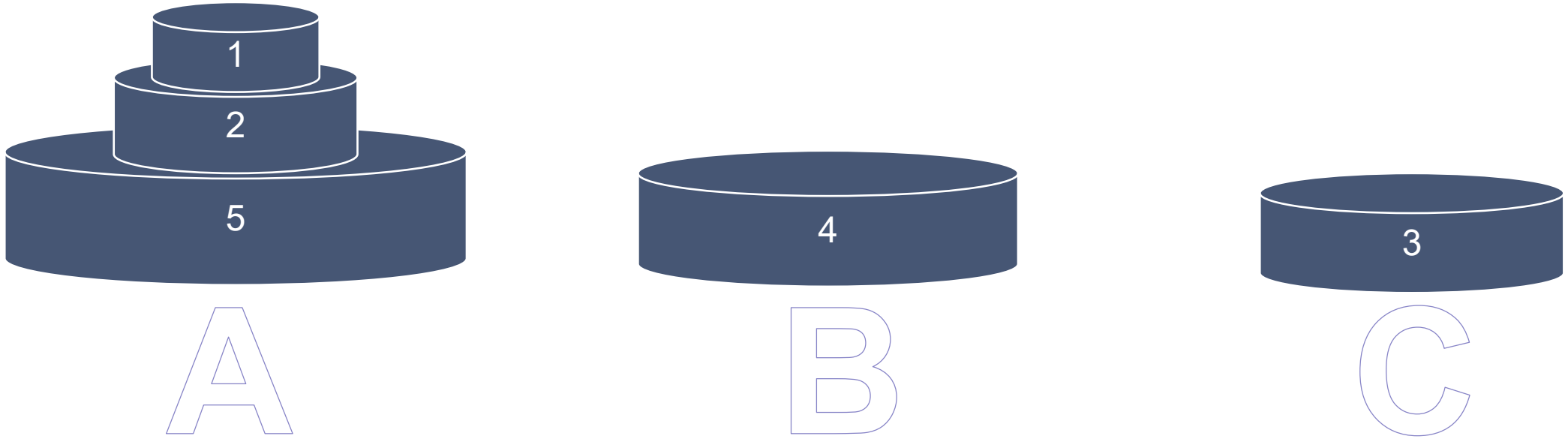
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



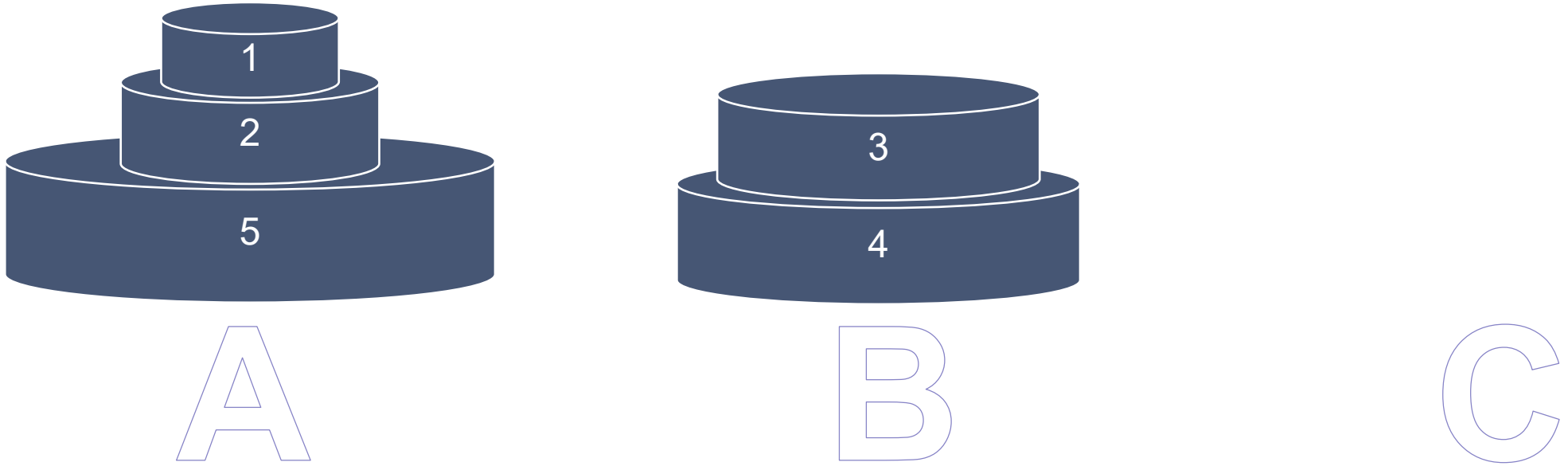
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



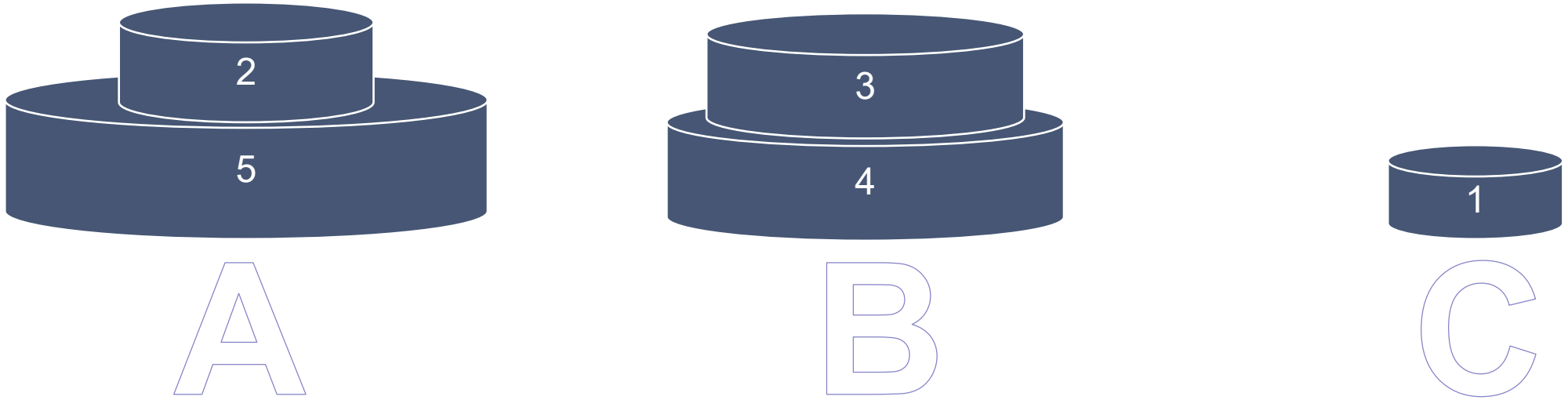
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



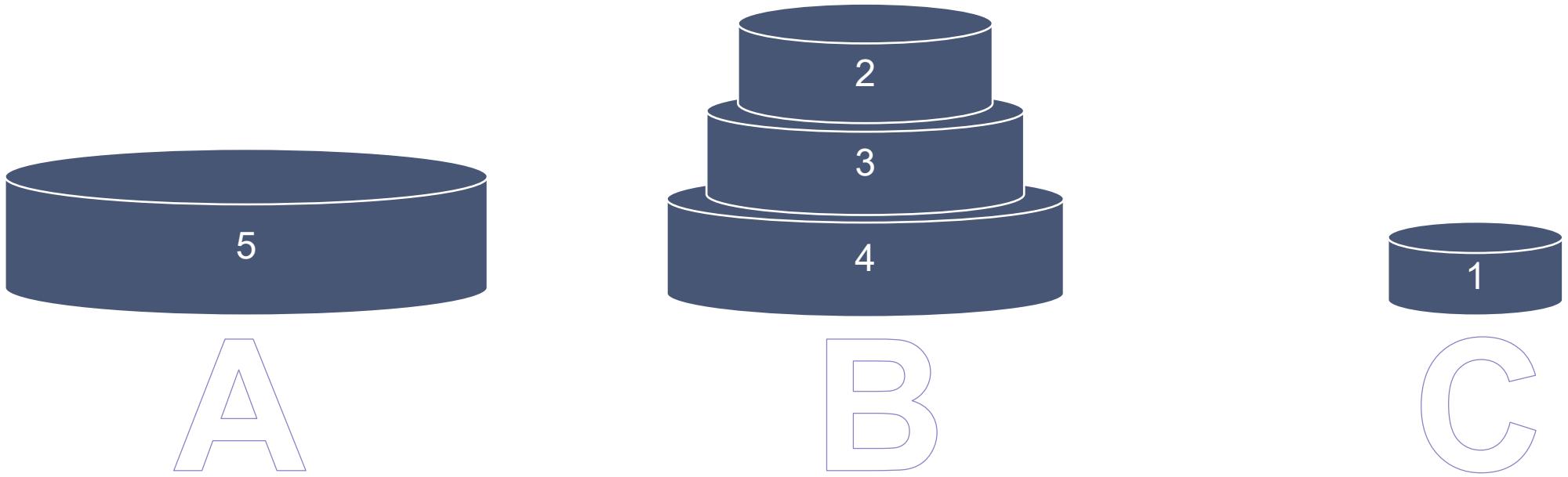
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



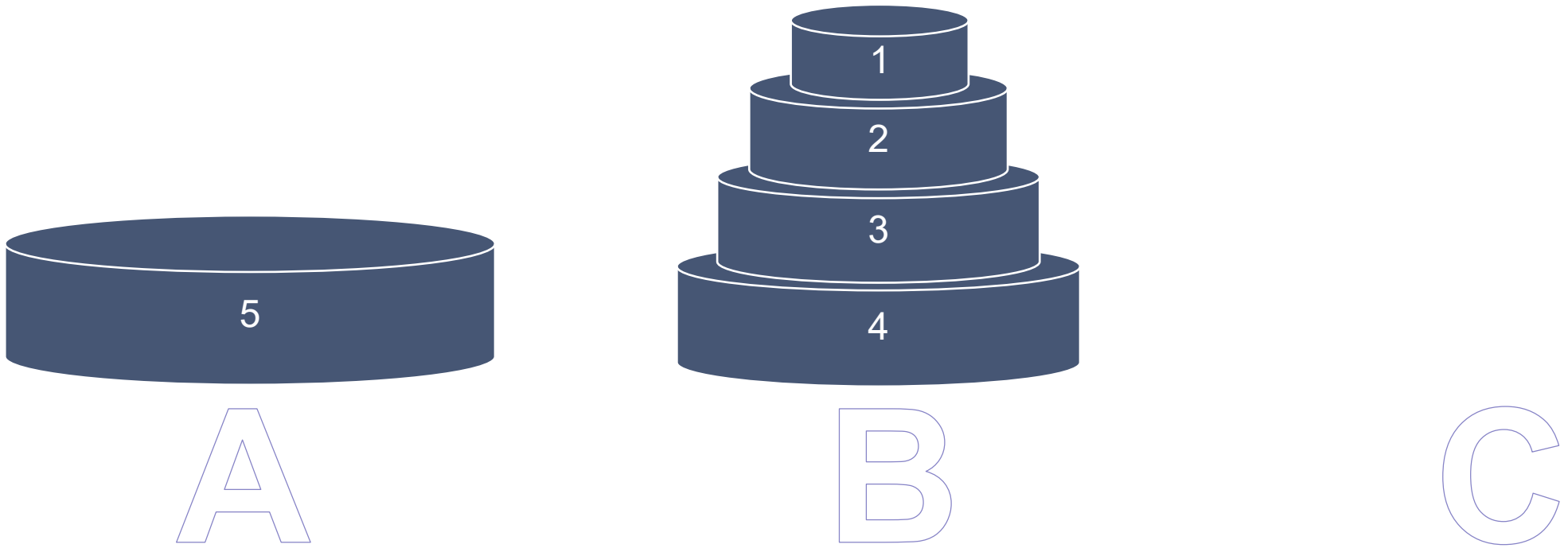
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



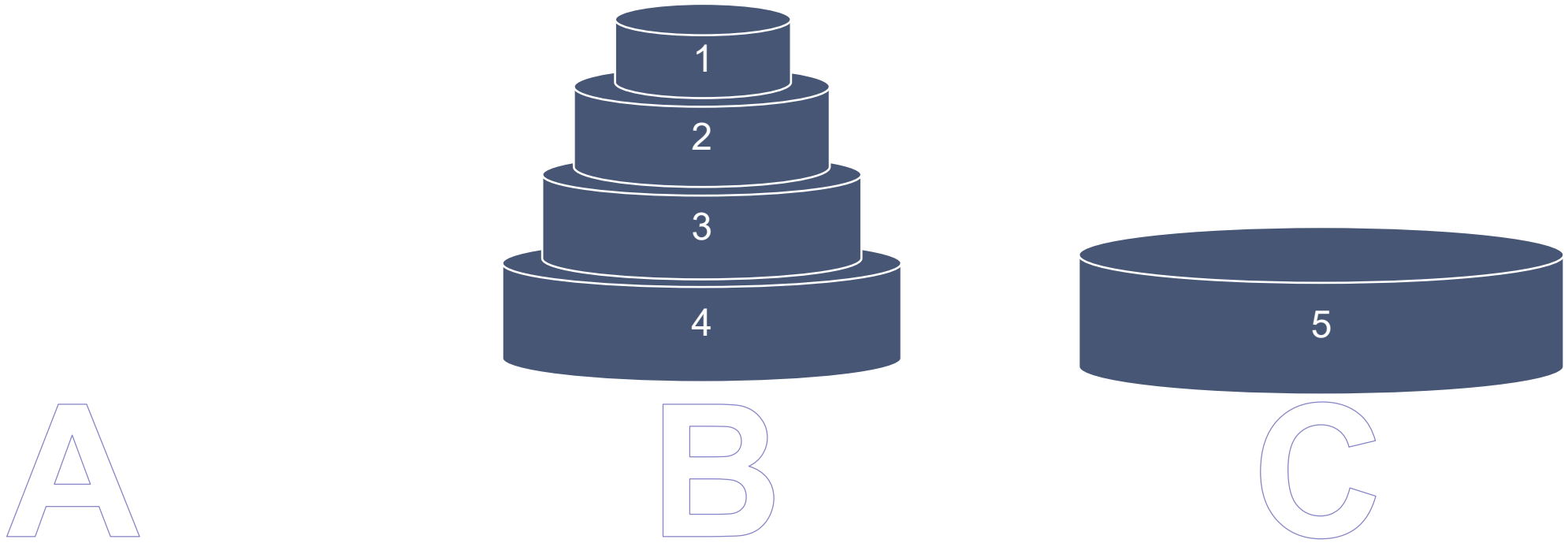
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



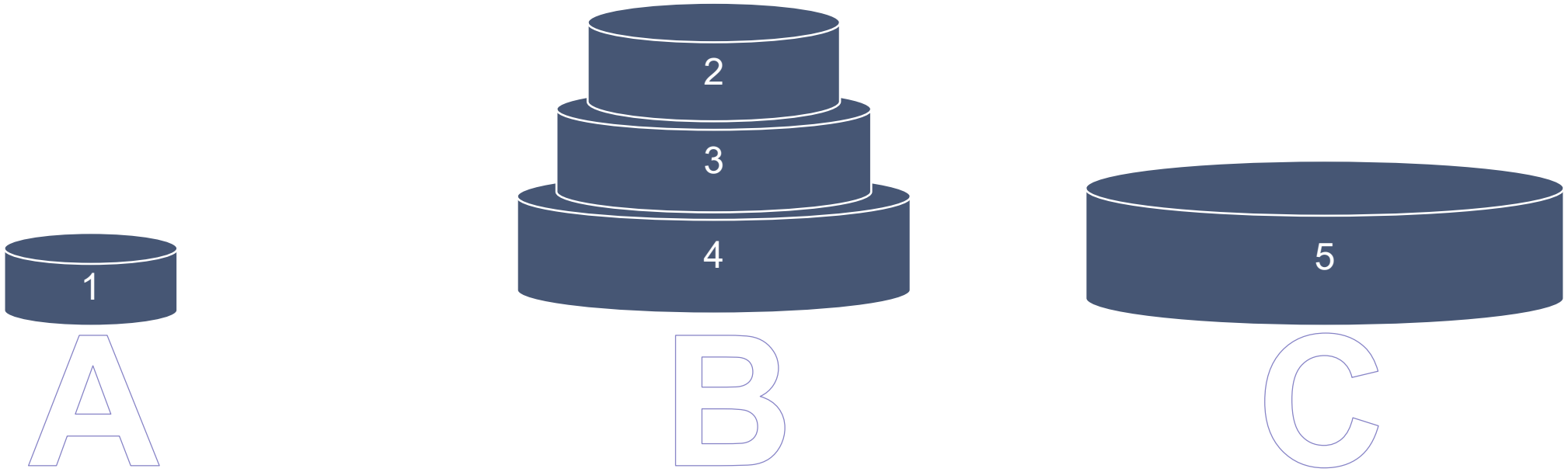
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



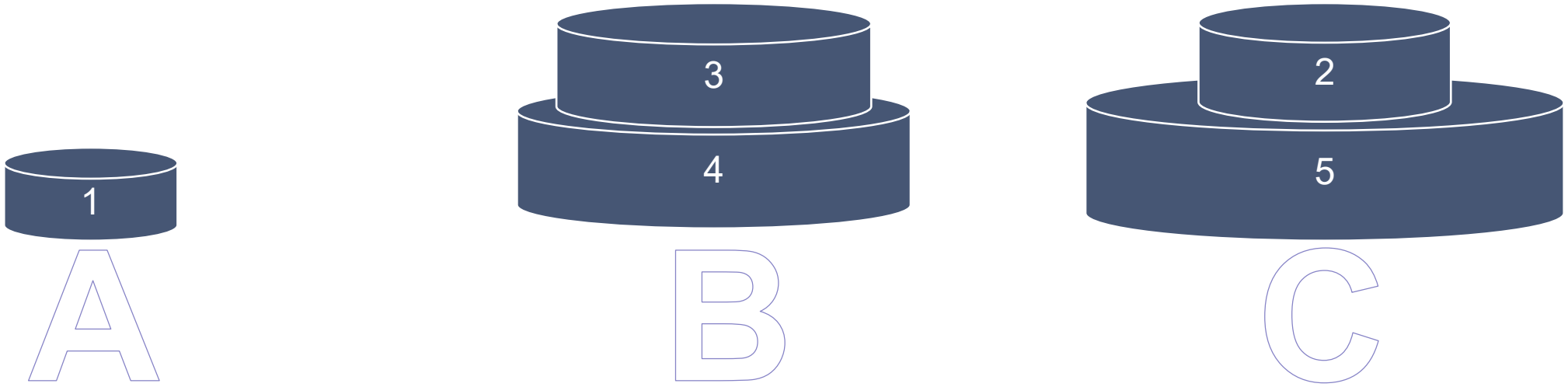
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



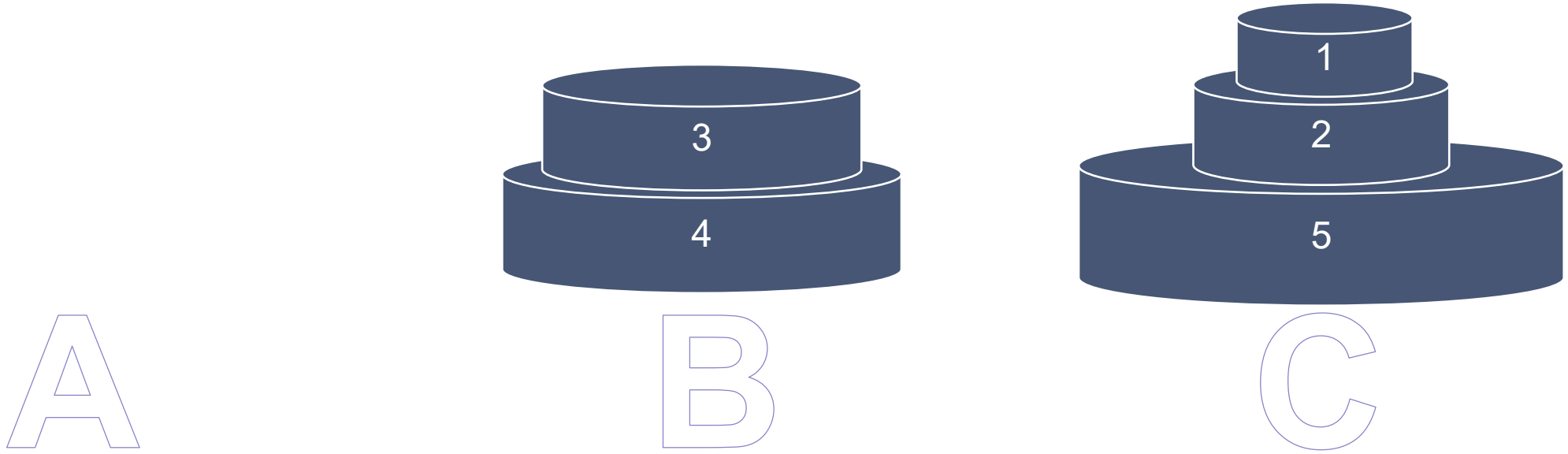
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



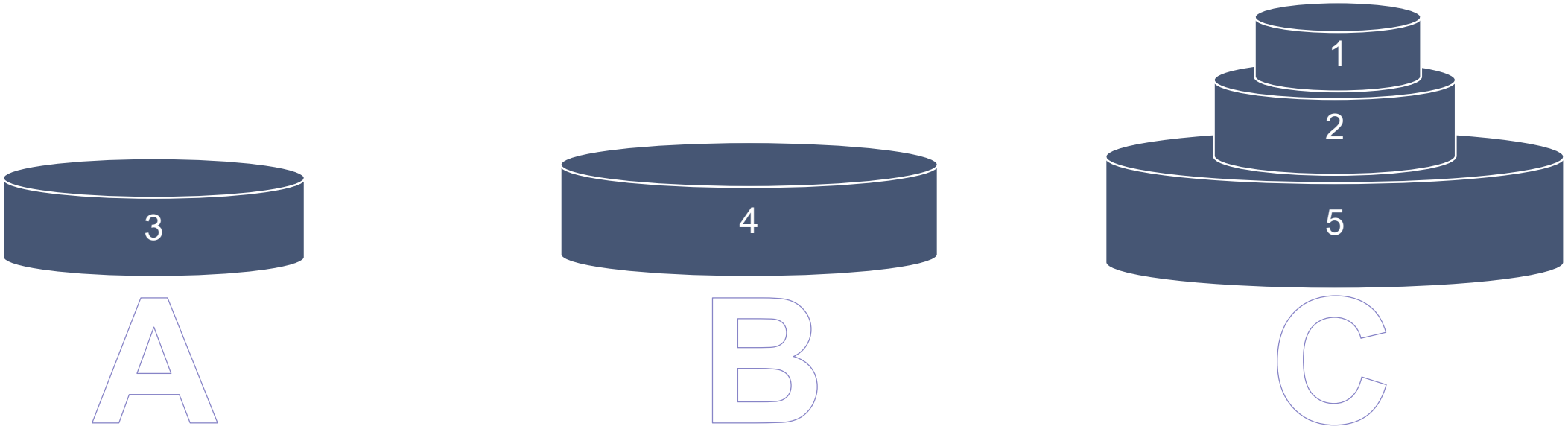
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



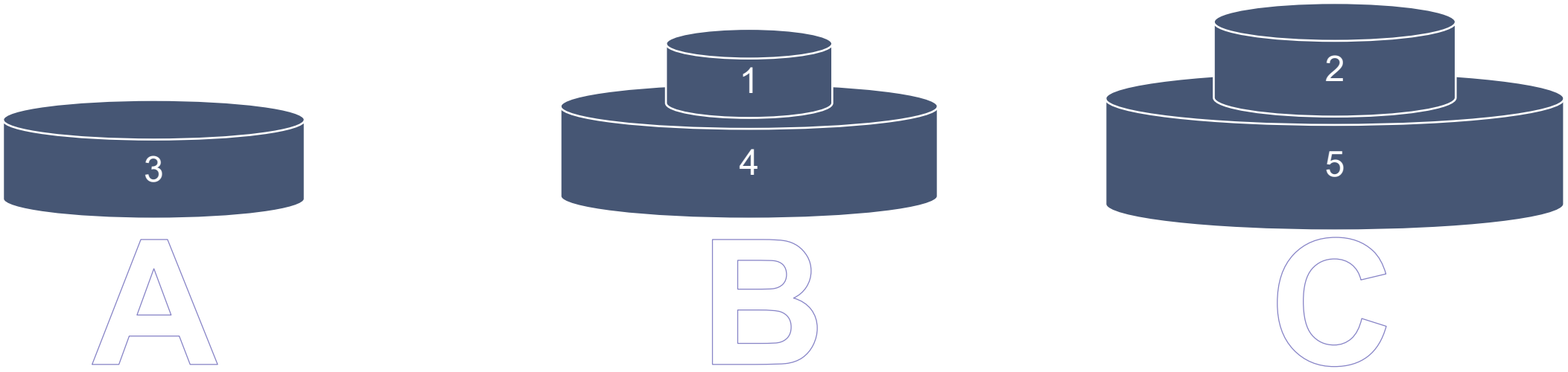
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



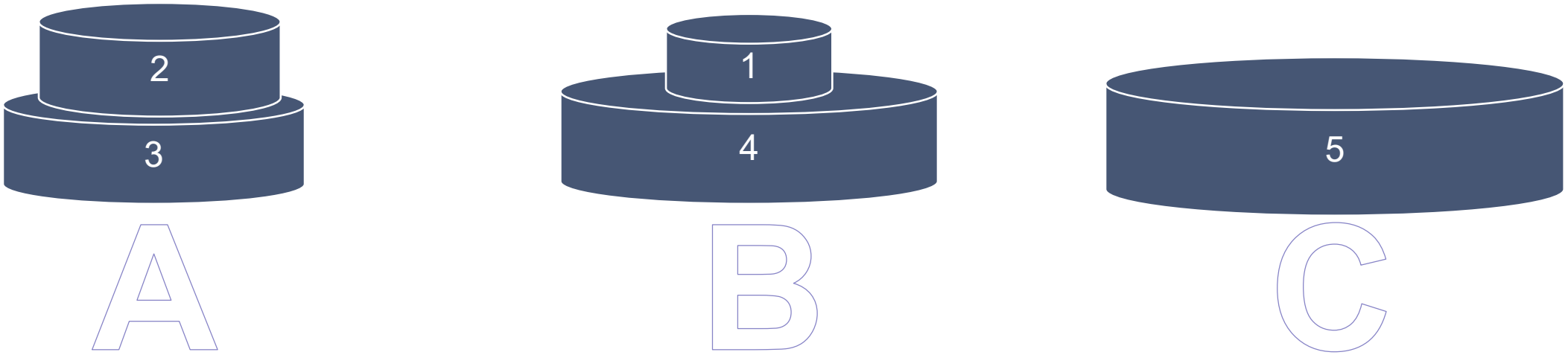
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



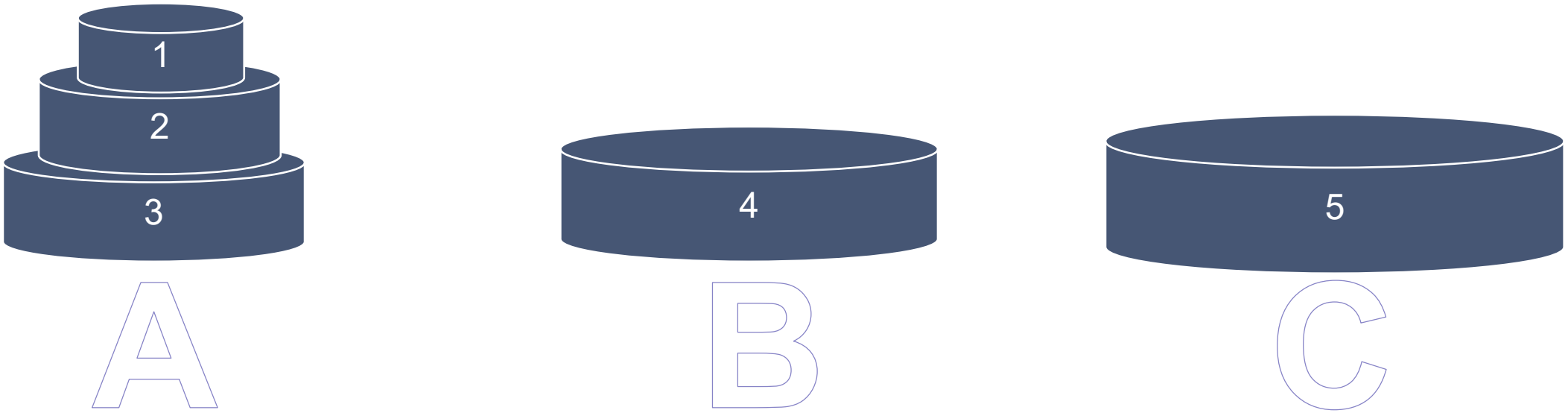
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



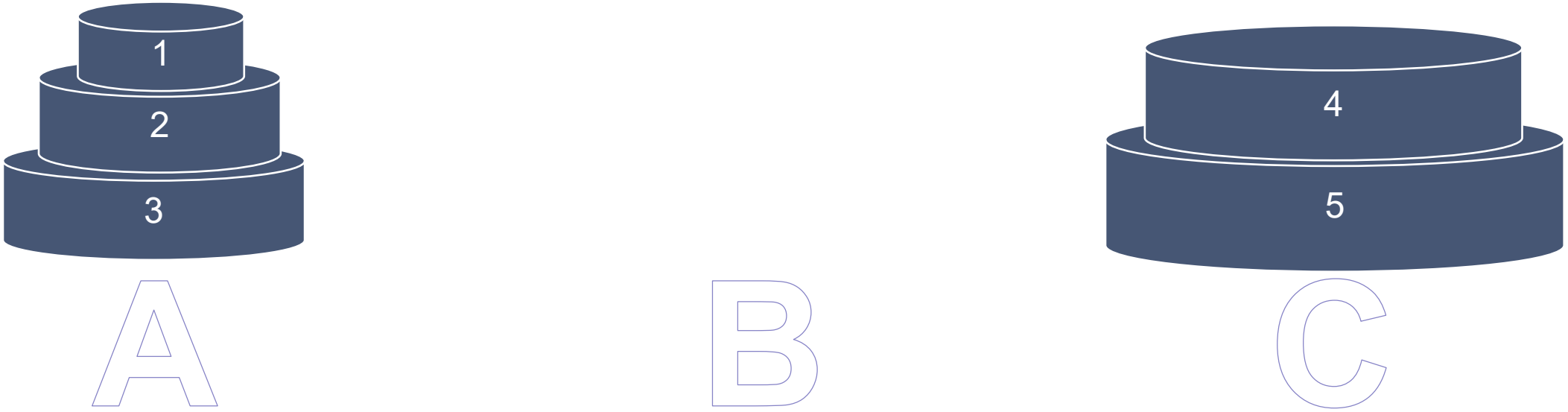
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



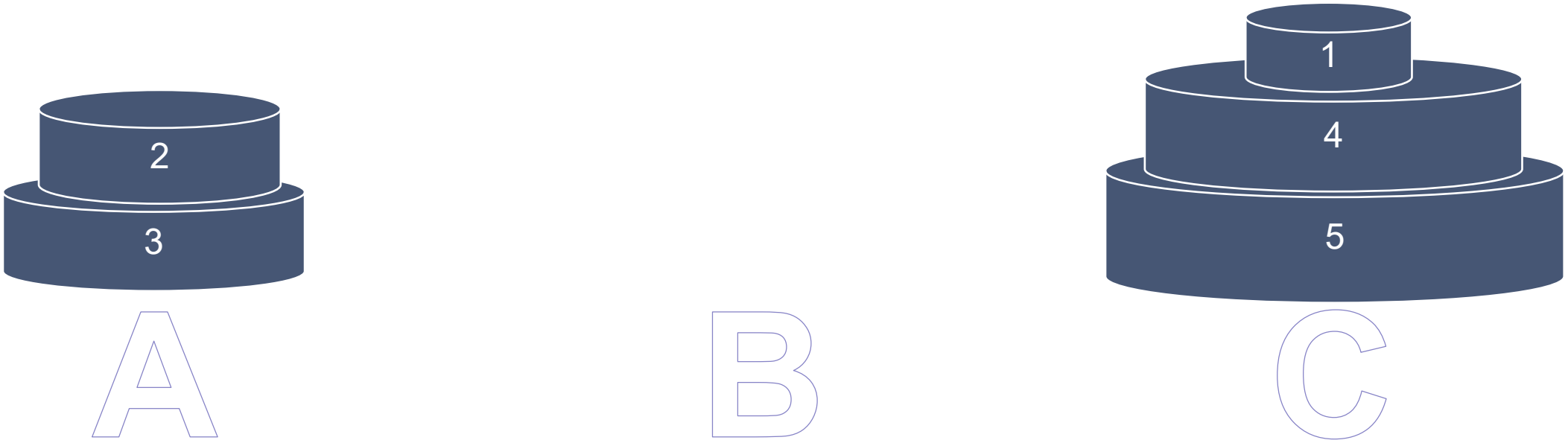
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



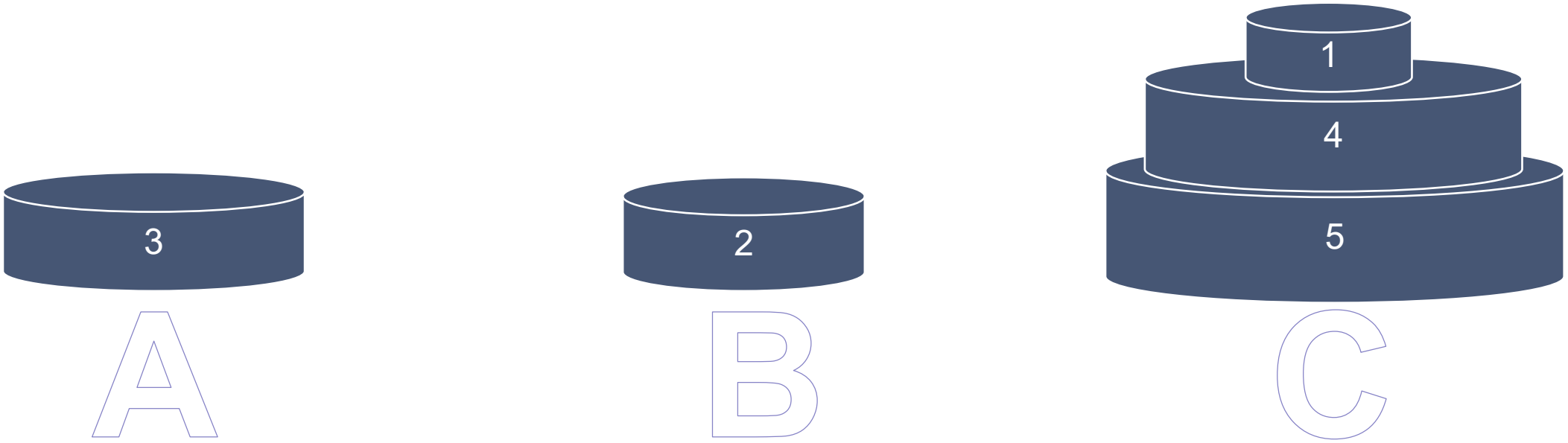
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



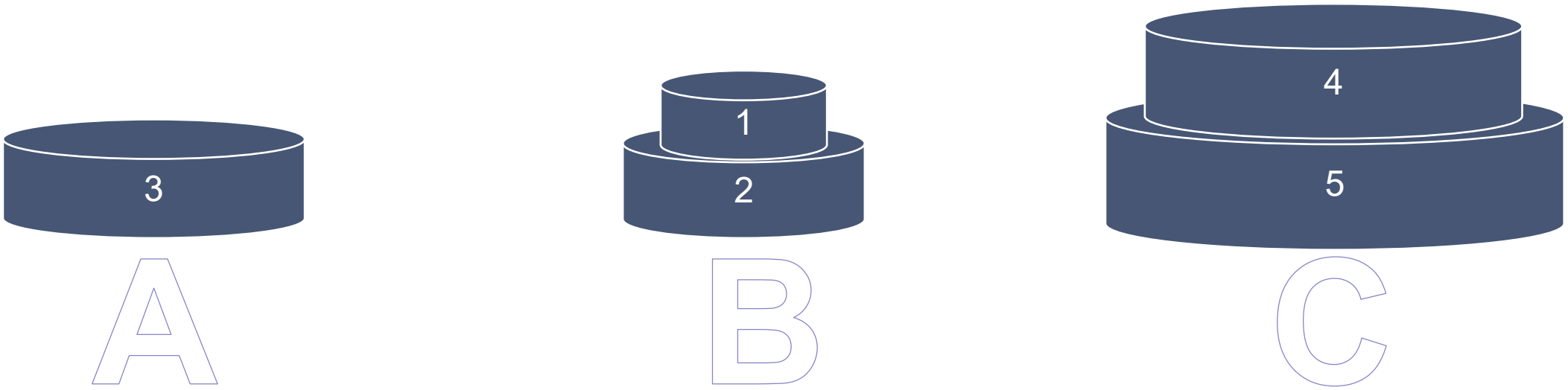
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



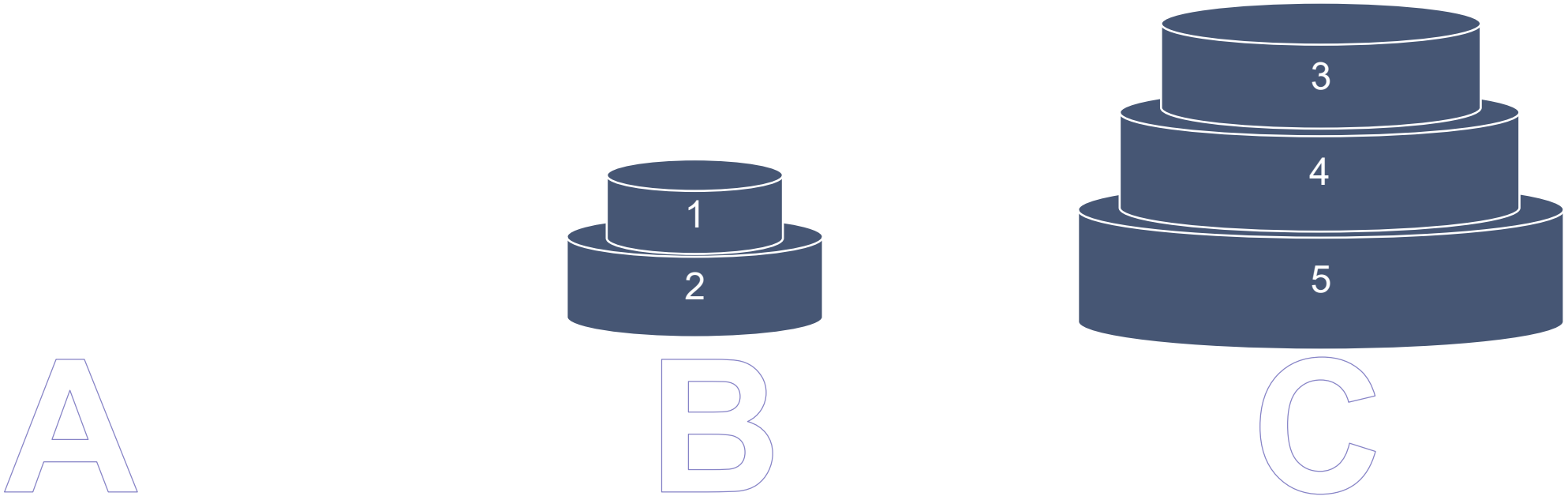
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



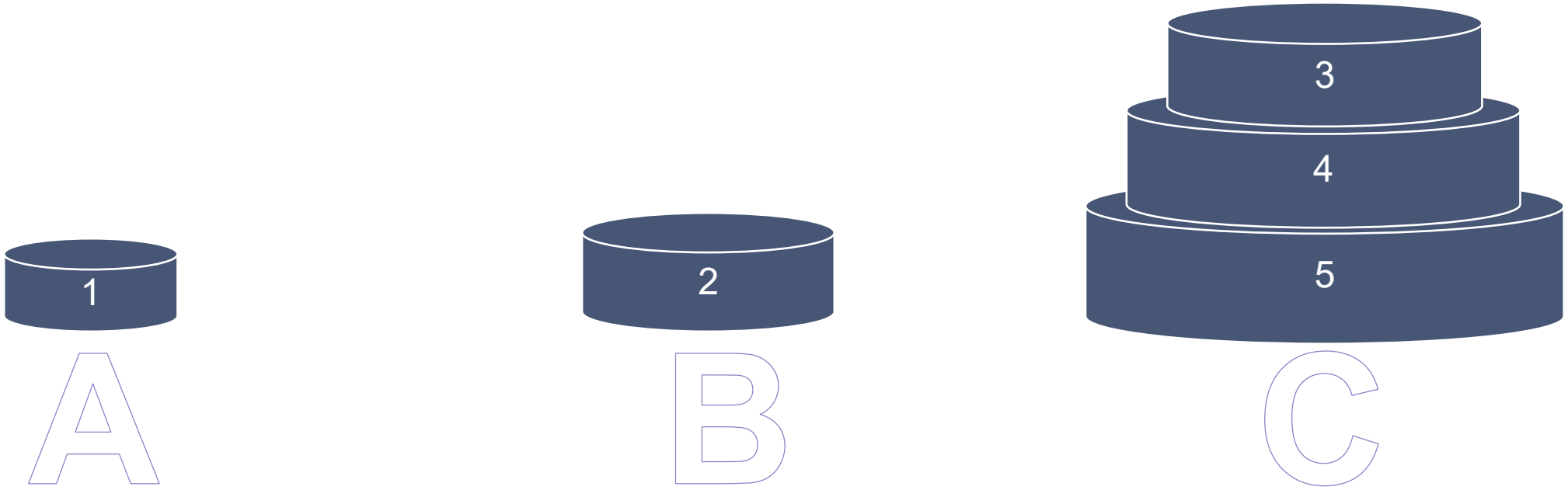
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



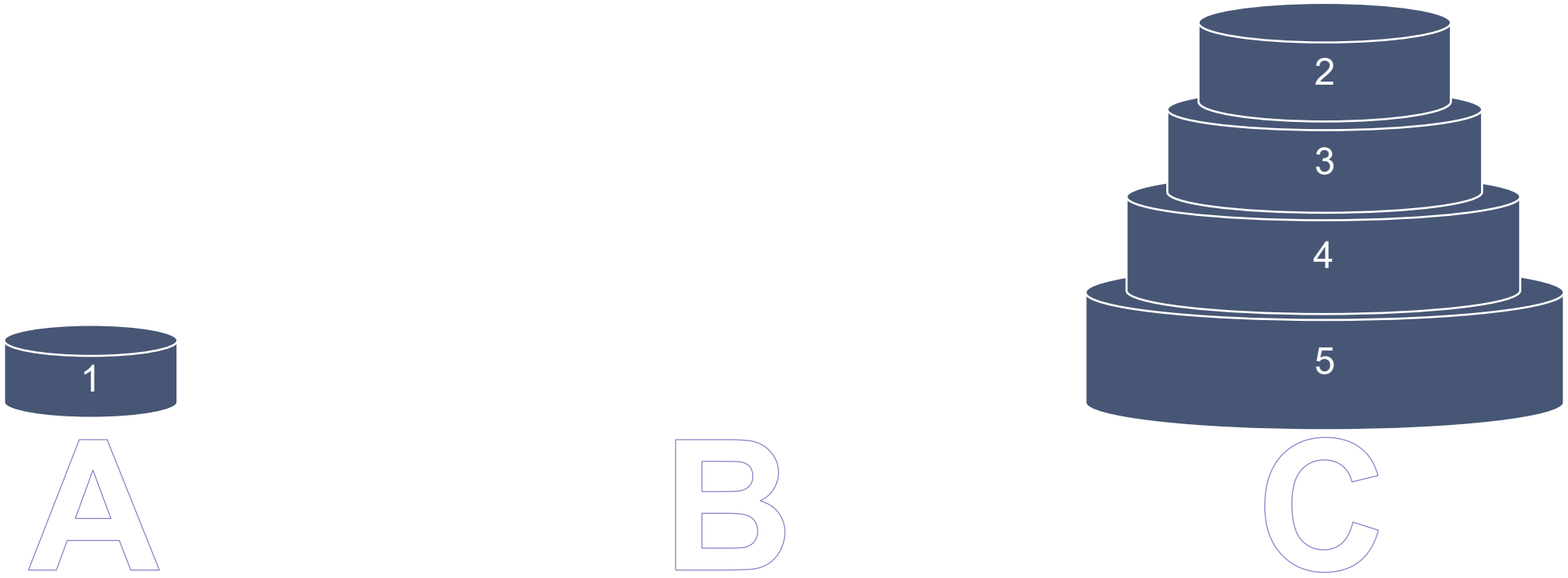
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



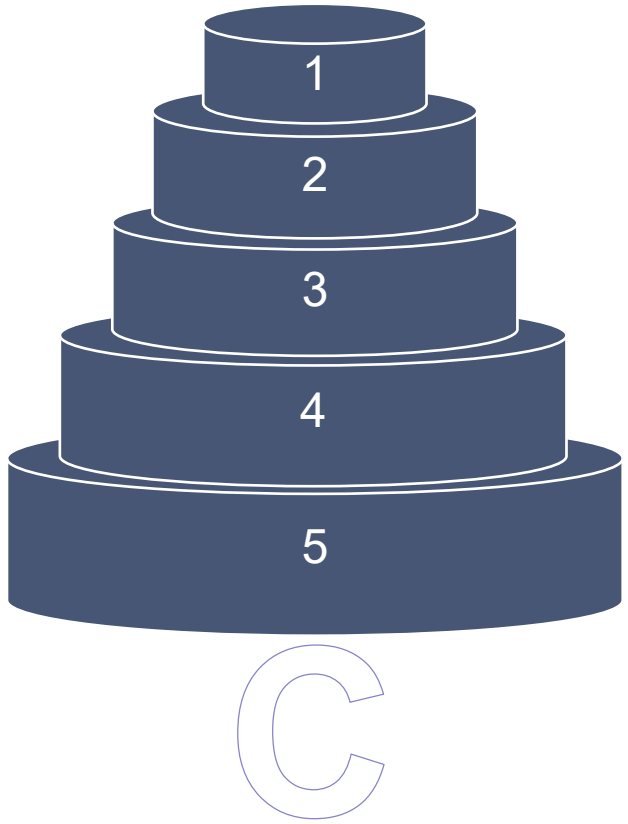
# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



# 3 NGĂN XẾP - STACK

**Minh họa bài toán tháp Hà Nội**



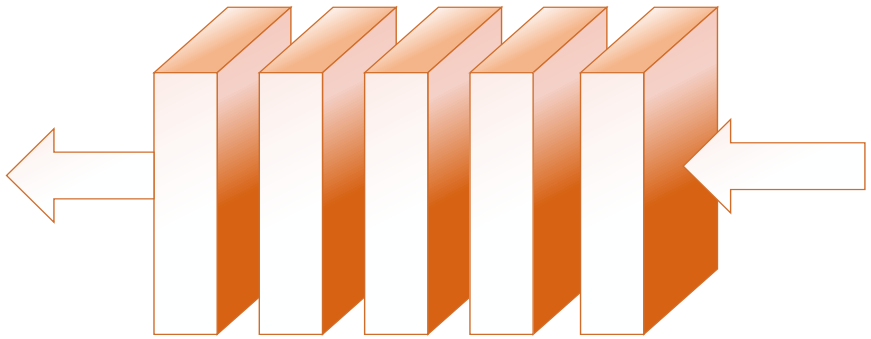
# 4 HÀNG ĐỢI - QUEUE

**Hàng đợi** (Queue): Là danh sách làm việc theo cơ chế FIFO (First In First Out), tức việc thêm 1 đối tượng vào hàng đợi hay lấy 1 đối tượng ra khỏi hàng đợi thực hiện theo cơ chế "vào trước ra trước".

Ví dụ tương tự hàng đợi là việc xếp hàng lên máy bay: Người nào xếp hàng trước thì lên máy bay trước tức là ra khỏi hàng đợi trước tiên.

Các thao tác trên hàng đợi bao gồm:

- Thêm đối tượng vào cuối hàng đợi.
- Lấy đối tượng từ đầu hàng đợi.
- Kiểm tra hàm đợi có rỗng hay không?
- Lấy giá trị của phần tử đầu của hàng đợi mà không hủy nó.



# 4 HÀNG ĐỢI - QUEUE

## Cài đặt hàng đợi bằng mảng:

```
#define max 100
//Cấu trúc dữ liệu của Queue
typedef struct tagQueue {
        int a[max];
        int Front; //Chỉ số của phần tử đầu trong Queue
        int Back; //Chỉ số của phần tử cuối trong Queue
} Queue;
//Khởi tạo Queue rỗng
void CreateQueue (Queue &q) {
        q.Front = -1;
        q.Back = -1;
}
```

# 4 HÀNG ĐỢI - QUEUE

## Cài đặt hàng đợi bằng mảng:

```
//Lấy 1 phần tử từ Queue
bool DeQueue (Queue &q, int &x) {
	if (q.Front != -1) {
		x = q.a[q.Front]; //Lấy ra phần tử đầu tiên của mảng
		q.Front++; //Thay đổi phần tử đầu tiên của Queue
		if (q.Front >q.Back) {//nếu Queue ban đầu chỉ có 1 phần tử
			q.Front = -1;
			q.Back = -1;
		}
		return true;
	} else {
		cout<<"Queue is empty";
		return false;
	}
}
```

# 4 HÀNG ĐỢI - QUEUE

## Cài đặt hàng đợi bằng mảng:

```
//Thêm 1 phần tử vào Queue
bool EnQueue (Queue &q, int x) {
        int i, f, b;
        if (q.Back - q.Front + 1 == max)//Queue đầy, không thể thêm
                return false;
        else {
                if (q.Front == -1){//Queue rỗng
                        q.Front = 0; q.Back =- 1;
                }
                if (q.Back == max-1){//Queue đầy
                        f = q.Front; b = q.Back;
                        for (i = f; i <= b; i++)
                                q.a[i-f] = q.a[i];//Reset lai các chỉ số cho mảng a
                        q.Front = 0; q.Back = b-f;
                }
                q.Back++; q.a[q.Back] = x; //Thêm phần tử vào cuối
                return true;
        }
}
```

# 4 HÀNG ĐỢI - QUEUE

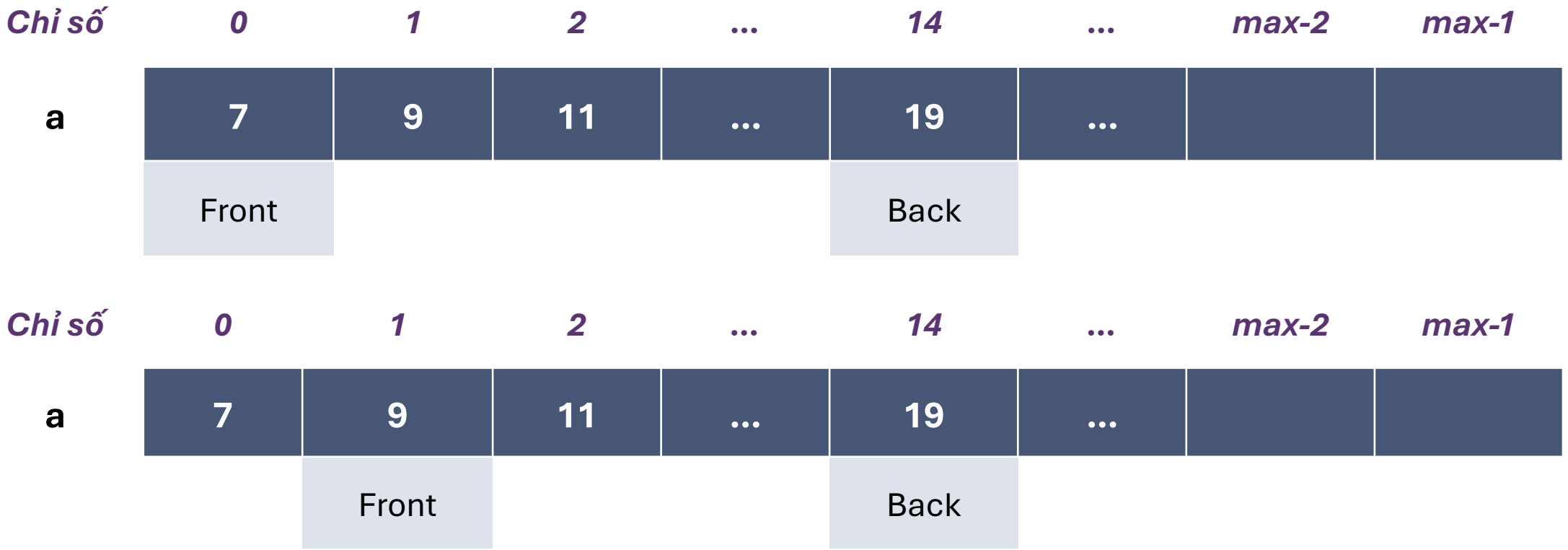
## Cài đặt hàng đợi bằng mảng:

```
//Thân chương trình
int main()
{
        Queue q;
        int i, x, t;
        CreateQueue(q);
        for(i=7; i<=20; i+=2)
                EnQueue (q , i);
        t = DeQueue (q, x);
        if(t)
                cout<<"Gia tri lay ra khoi hang doi la: "<<x;
                //In ra 7
}
```

# 4 HÀNG ĐỢI - QUEUE

## Cài đặt hàng đợi bằng mảng:

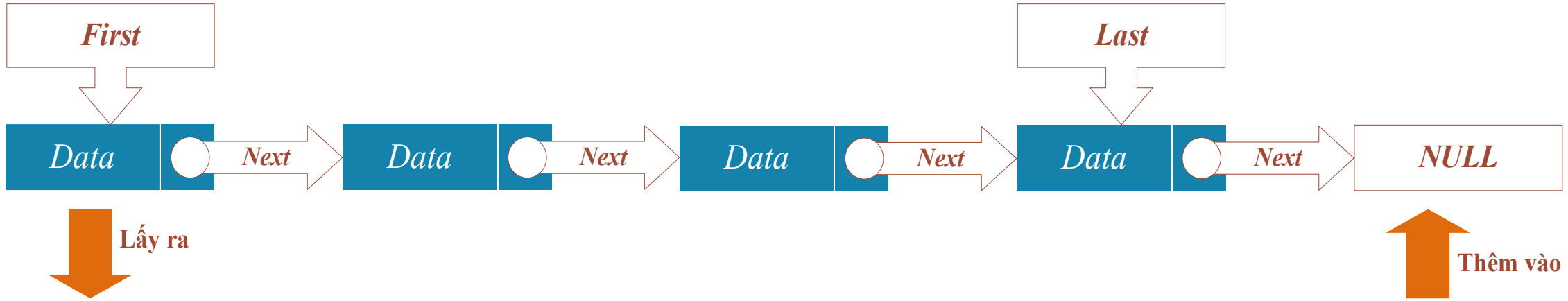
Minh họa lấy 1 phần tử ra khỏi hàng đợi



# 4 HÀNG ĐỢI - QUEUE

## Cài đặt hàng đợi bằng danh sách liên kết:

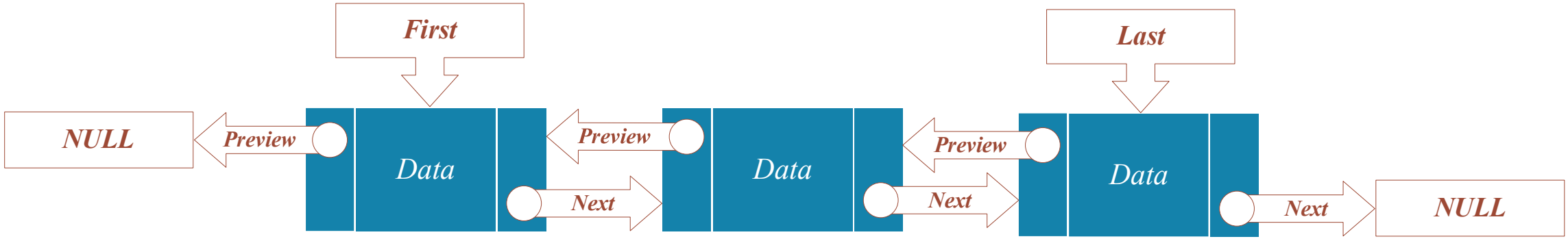
Để cài đặt hàng đợi bằng DSLK, ta cũng sử dụng 1 DSLK đơn với 2 con trỏ ***first** và ***last** để lưu trữ nút đầu và cuối của danh sách. Các thao tác thêm vào và lấy ra ta sẽ thực hiện theo thứ tự thêm vào ở cuối và lấy ra ở đầu của DSLK đơn.



# 5 MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

## 1. Danh sách liên kết kép

Danh sách liên kết kép (doubly linked list) là danh sách liên kết mà mỗi nút trong đó sẽ liên kết với cả nút đằng sau (next) và nút đằng trước nó (preview).



# 5 MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

## 1. Danh sách liên kết kép

Cấu trúc dữ liệu của 1 nút trong DSLK kép:

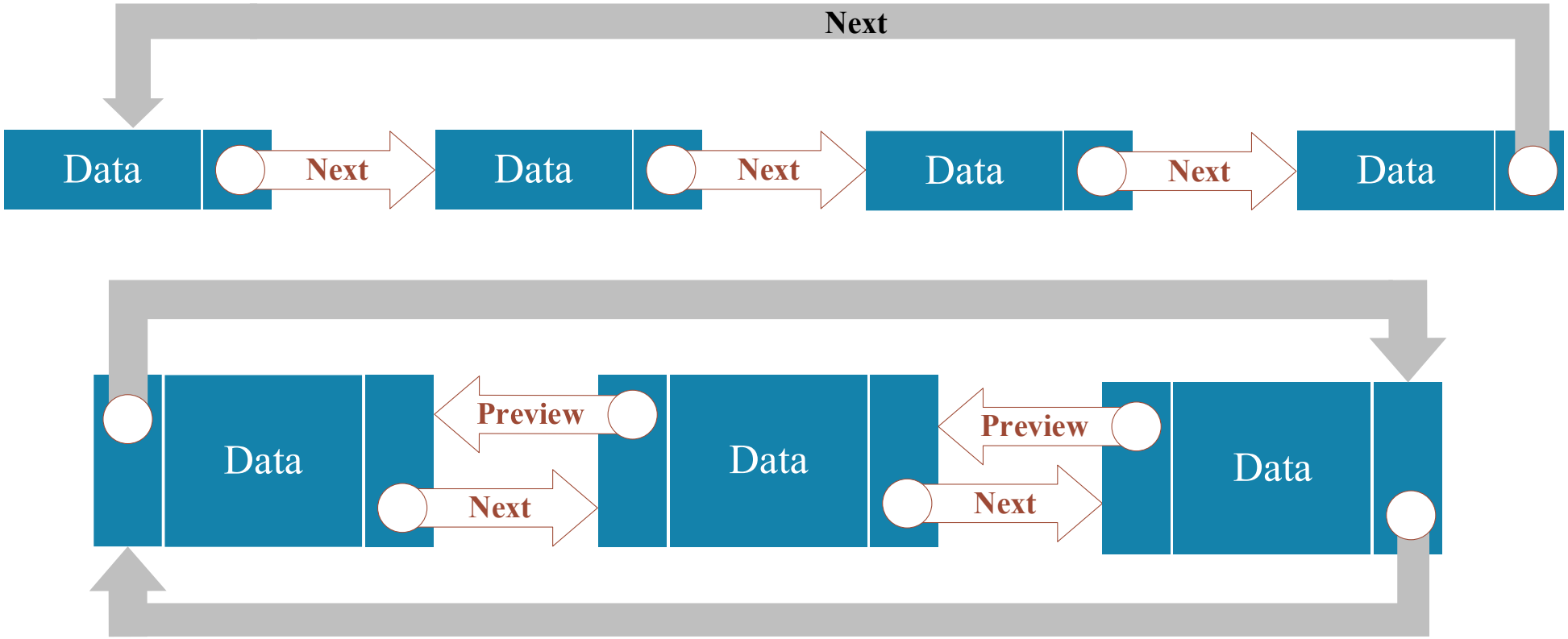
```
struct Node {
        int data;
        Node *next;
        Node *preview;
};
struct doubleList {
        Node *first;
        Node *last;
};
```

Dựa vào cấu trúc nút ở trên ta cũng có thể dễ dàng thực hiện các thao tác đối với danh sách liên kết kép tương tự như là với danh sách liên kết đơn.

# 5 MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

## 2. Danh sách liên kết vòng

Danh sách liên kết vòng là danh sách liên kết trong đó nút cuối cùng sẽ liên kết với nút đầu tiên.



# BÀI TẬP CHƯƠNG 4

**Bài 1**. Sử dụng kiểu dữ liệu danh sách liên kết đơn, viết chương trình quản lý danh sách sinh viên, thông tin mỗi sinh viên bao gồm: mã số, họ tên, năm sinh, điểm trung bình. Thực hiện các yêu cầu sau:

a) Khai báo, khởi tạo, và nhập danh sách.

b) Duyệt và in danh sách.

c) Tìm kiếm một sinh viên theo họ tên.

d) Chèn thêm một sinh viên mới vào sau một sinh viên có mã số chỉ định.

e) Xóa một sinh viên có mã sinh viên chỉ định.

f) Sắp xếp danh sách theo mã số sinh viên tăng dần.

g) Sắp xếp danh sách theo tên sinh viên theo thứ tự bảng chữ cái A, B, C, ….

h) Hủy danh sách khi kết thúc chương trình.

# BÀI TẬP CHƯƠNG 4

**Bài 2**. Xây dựng cấu trúc dữ liệu stack cài đặt bằng danh sách liên kết để đổi số hệ 10 sang các hệ cơ số khác. Hãy xây dựng chuỗi kết quả là số n ở các hệ cơ số sau:

a) Hệ nhị phân.

b) Hệ bát phân.

c) Hệ thập lục phân.

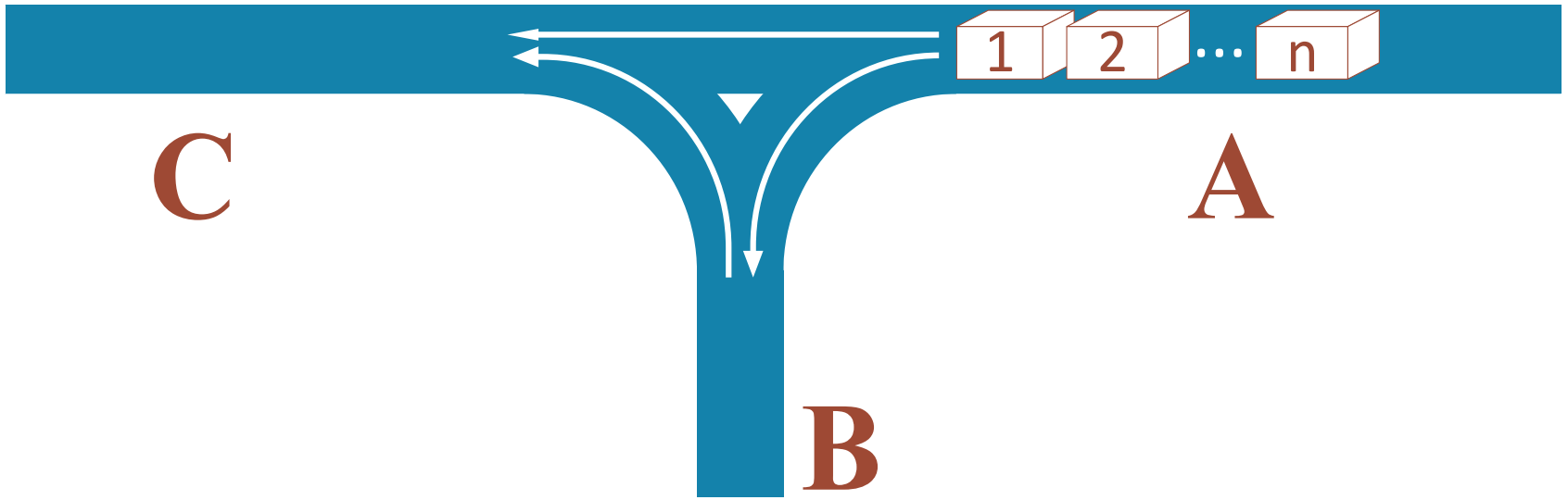
# BÀI TẬP CHƯƠNG 4

**Bài 3**. Bài toán di chuyển toa tàu (hình dưới):

Các toa được đánh số từ 1 đến n, đường di chuyển có thể là các vạch mũi tên.

Ta cần di chuyển các toa từ A đến C sao cho tại C các toa tàu được sắp xếp các thứ tự mới nào đó.

Hãy nhập vào thứ tự tại C cần có, cho biết có cách chuyển không? Nếu có, hãy trình bày cách chuyển.



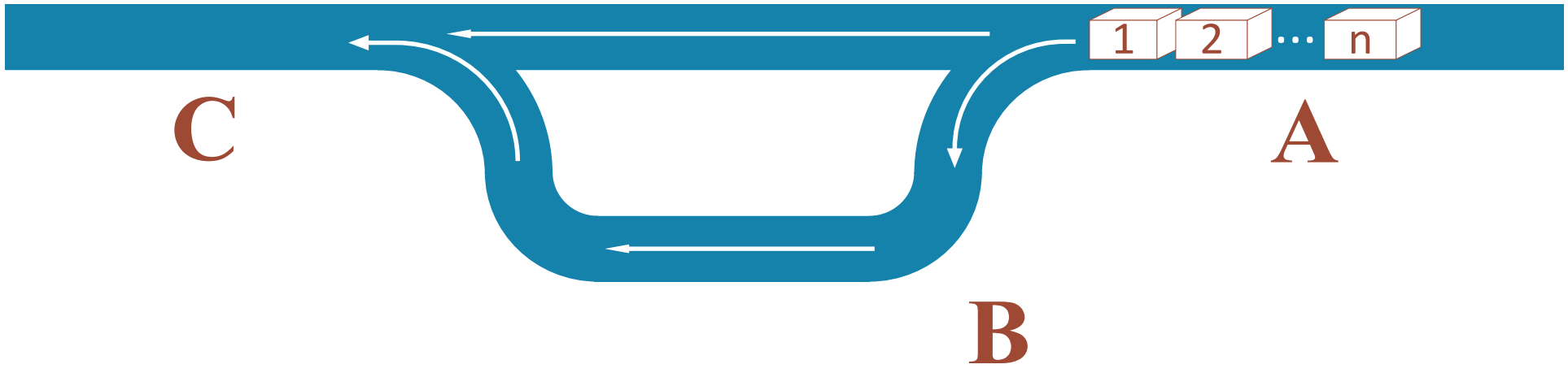
# BÀI TẬP CHƯƠNG 4

**Bài 4**. Bài toán di chuyển toa tàu hình dưới.

Các toa được đánh số từ 1 đến n, đường di chuyển có thể là các đường theo hướng mũi tên.

Ta cần di chuyển các toa từ A tới C sao cho tại C các toa tàu được sắp xếp theo các thứ tự mới nào đó.

Hãy nhập vào thứ tự tại C cần có, cho biết có cách chuyển không? Nếu có, hãy trình bày cách chuyển.



# BÀI TẬP CHƯƠNG 4

**Bài 5**. Xây dựng CTDL queue cài đặt bằng DSLK đơn để mô phỏng qui trình cho thuê máy ở một tiệm NET với các đặc tả như sau:

- Mỗi máy trong tiệm NET sẽ được đánh số khác nhau để dễ phân biệt và quản lý.
- Các máy còn trống (chưa có người thuê) sẽ được lưu trữ trong hàng đợi A. Mỗi nút của hàng đợi A sẽ lưu trữ thông tin số máy.
- Danh sách khách hàng chờ nhận máy sẽ được lưu trữ trong hàng đợi Q. Mỗi nút của hàng đợi Q sẽ chứa họ tên của khách hàng.
- Danh sách khách hàng đang thuê máy sẽ được quản lý bằng DSLK đơn H. Mỗi nút của H sẽ chứa thông tin họ tên người thuê máy, thời gian bắt đầu thuê máy, số máy.

Viết chương trình quản lý phòng NET với các chức năng chính như sau: (Trang sau)

# BÀI TẬP CHƯƠNG 4

**Bài 5.** (TT)

**a) Đăng ký thuê máy:**

- Nếu còn máy trống trong A thì lấy máy trong A cho khách thuê, sau đó tạo nút mới với thông tin khách vừa thuê và thêm vào H. Lấy máy vừa cho thuê ra khỏi A.
- Nếu không còn máy sẵn sàng trong A thì thêm khách vào hàng đợi Q để chờ có máy trống.

**b) Khách hàng trả máy:**

- Xóa nút chứa thông tin khách trả máy khỏi H.
- Thêm nút mới với thông tin máy vừa được trả vào A.

**c) Phục vụ khách chờ:** Kiểm tra A và Q để xem có khách hàng đang chờ và còn máy trống hay không? Nếu có, tiến hành cho khách thuê và cập nhật lại A, Q, H.

# BÀI TẬP CHƯƠNG 4

**Bài 6**. Cài đặt 1 DSLK đơn biểu diễn thông tin cho n thành phố. Thông tin các thành phố gồm: Tên, diện tích, dân số.

a) Nhập/xuất dữ liệu cho DSLK.

b) Tìm thông tin thành phố có diện tích lớn nhất và thêm vào sau nó 1 thành phố.

c) Xóa khỏi DSLK thành phố “Ha Noi” nếu có.

d) Sắp xếp DSLK theo chiều tăng dần của dân số.

# BÀI TẬP CHƯƠNG 4

**Bài 7**. Sử dụng cấu trúc dữ liệu danh sách liên kết đôi vòng để viết chương trình quản lý các chuyến bay của một công ty hàng không, mỗi chuyến bay gồm: Mã chuyến, ngày, giờ khởi hành, điểm đến. Thực hiện các yêu cầu sau:

a) Khai báo và khởi tạo danh sách list.

b) Nhập danh sách bằng cách thêm vào list ở vị trí phù hợp để danh sách có thứ tự tăng của mã chuyến.

c) Hãy in tất cả các chuyến bay khởi hành trong ngày chỉ định.

d) Hủy danh sách khi kết thúc chương trình.