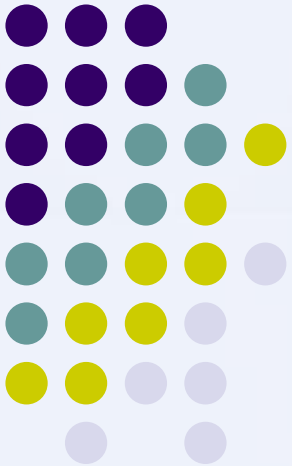
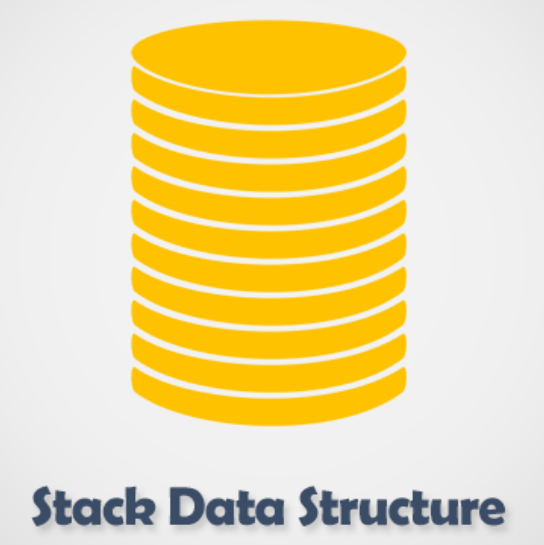
# Bài 8. Cấu trúc dữ liệu ngăn xếp



# Stack




❖Stack là cách tổ chức lưu trữ các đối tượng dưới dạng một danh sách tuyến tính mà việc bổ sung đối tượng và lấy các đối tượng ra được thực hiện ở cùng một đầu của danh sách.

❖Stack được gọi là danh sách kiểu LIFO (Last In First Out - vào sau ra trước)

# Các vấn đề cần nghiên cứu

- Cấu trúc dữ liệu trừu tượng Stack (ADT Stack)
- Những ứng dụng của Stack
- Cài đặt Stack dựa trên mảng
- Sự phát triển stack dựa trên mảng

# Cấu trúc dữ liệu trừu tượng (ADT- Abstract Data Type)

- **Các thành phần của một ADT**
  - Dữ liệu được lưu trữ
  - Các phép toán trên dữ liệu
  - Các điều kiện xảy ra lỗi kết hợp với các phép toán

- **Ví dụ:** Mô hình ADT của một hệ thống kho hàng đơn giản

  - Dữ liệu được lưu trữ theo phiếu mua/bán
  - Các phép toán:
    - + Hóa đơn buy (kho, số lượng, giá)
    - + Hóa đơn sell (kho, số lượng, giá)
    - + Hủy cancel (Số hóa đơn) //Số hóa đơn

  Điều kiện lỗi:

  - Mua/bán một mặt hàng không có trong kho
  - Hủy bỏ một phiếu mà phiếu không tồn tại

# Cấu trúc dữ liệu trừu tượng Stack

- Stack ADT lưu trữ các đối tượng bất kỳ
- Bổ sung và lấy ra các phần tử theo kiểu "Vào sau ra trước" – "Last In First Out"
- Các phép toán chính:

  push(Object o): bổ sung đối tượng o vào Stack

  pop(): lấy ra và trả lại phần tử được bổ sung vào cuối cùng của Stack

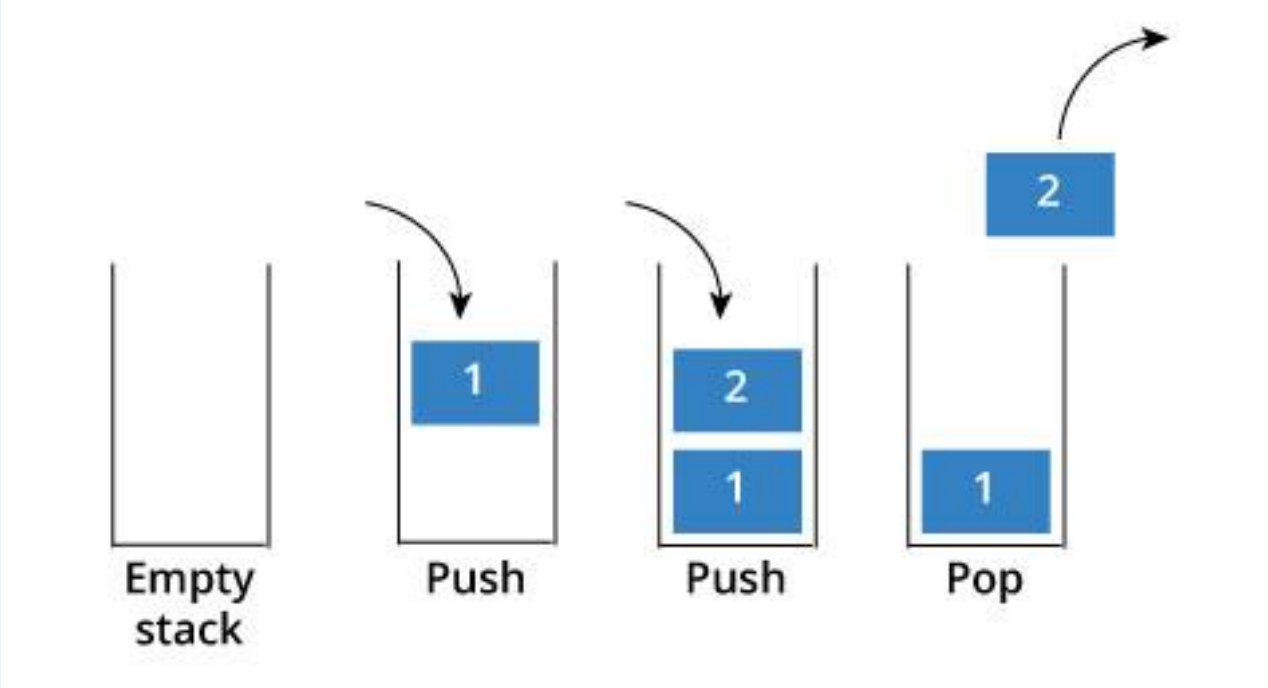
- Các phép toán bổ trợ

  top() trả lại tham chiếu đến phần tử được bổ sung vào cuối cùng của Stack

  size(): trả lại số phần tử hiện lưu trữ trong Stack

  empty(): trả lại giá trị kiểu boolean để xác định Stack có lưu trữ phần tử nào hay không

# Cấu trúc dữ liệu trừu tượng Stack



# Các trường hợp ngoại lệ

- Ngoại lệ: là việc thực hiện một phép toán mà trong trường hợp đó nó không thể thực hiện
- Với Stack ADT thì phép toán pop và top không thể thực hiện được nếu Stack rỗng

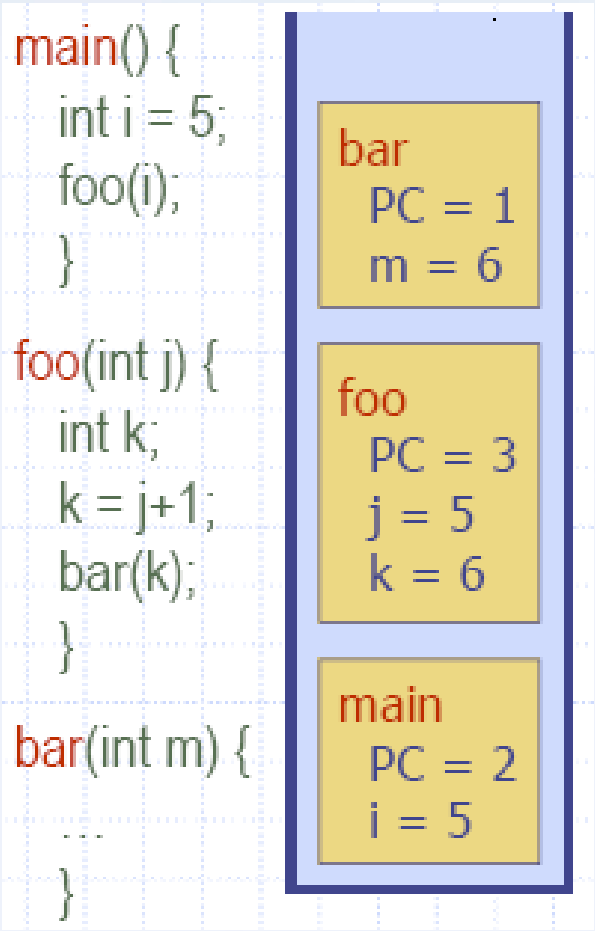
Khi thực hiện phép toán pop hoặc top trên một Stack rỗng thi dẫn đến ngoại lệ Stack rỗng

# Một số ứng dụng của Stack

- Các ứng dụng trực tiếp
  - Lưu lại các trang Web đã thăm trong một trình duyệt
  - Thứ tự Undo trong một trình soạn thảo
  - Lưu trữ các biến khi một hàm gọi tới hàm khác, và hàm được gọi lại gọi tới hàm khác, và cứ tiếp tục như vậy.
- Các ứng dụng gián tiếp
  - Cấu trúc dữ liệu bổ trợ cho một số thuật toán
  - Là một thành phần của những cấu trúc dữ liệu khác

# Ví dụ: Sự thực hiện trong hệ thống được viết bằng C++

- Hệ thống được viết bằng C++ khi chạy sẽ giữ các phần của một chuỗi mắt xích của các hàm đang hoạt động trong một Stack
- Khi một hàm được gọi, hệ thực hiện đẩy vào Stack một khung chứa bao gồm:
  - Các biến cục bộ và giá trị trả lại của hàm
- Khi một hàm trả lại giá trị, cái khung của nó trong Stack sẽ được lấy ra và máy sẽ tiếp tục thực hiện đến phương thức ở đỉnh của Stack



# Cài đặt Stack bằng mảng

- Cách đơn giản nhất cài đặt một Stack là sử dụng một mảng
- Chúng ta thực hiện bổ sung phần tử vào từ trái qua phải
- Sử dụng một biến *t* lưu chỉ số của phần tử ở đỉnh của Stack

```
Algorithm size()
  return t + 1

Algorithm pop()
  if isEmpty() then
    throw EmptyStackException
  else
    t ← t − 1
    return S[t + 1]
```

# Cài đặt Stack bằng mảng (tiếp)

- Mảng lưu trữ các phần tử của Stack có thể dẫn đến đầy
- Phép toán bổ sung các phần tử có thể dẫn đến ngoại lệ: *FullStackException*

  - Giới hạn của mảng được sử dụng cài đặt

  - Không phải là bản chất của Stack ADT

```
Algorithm push(o)
    if t = S.length − 1 then
        throw FullStackException
    else
        t ← t + 1
        S[t] ← o
```

# Thực hành và những hạn chế

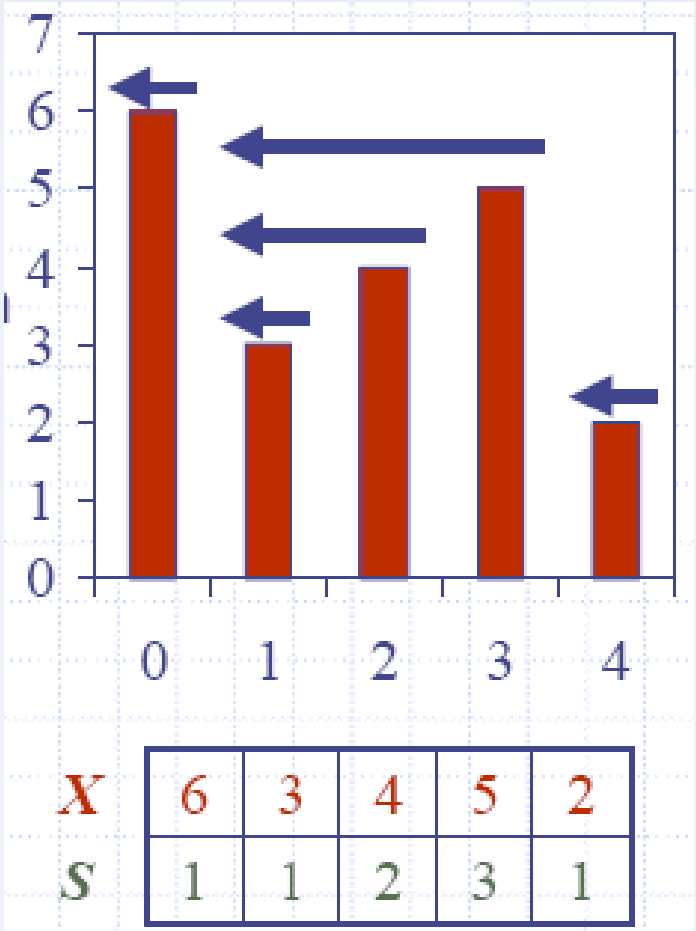
- **Thực hành**
  - Cho n là số phần tử của Stack
  - Không gian cần sử dụng là O(n)
  - Mỗi một phép toán chạy trong thời gian O(1)
- **Những hạn chế**
  - Kích thước tối đa của Stack phải định nghĩa trước, và không thể thay đổi được
  - Cố gắng bổ sung một phần tử vào Stack khi Stack đầy sẽ dẫn đến ngoại lệ

# Bài tập

- Cài đặt Stack bằng mảng
- Xây dựng một chương trình ứng dụng stack với các chức năng sau:
  1. Thêm một phần tử vào stack
  2. Lấy một phần tử ra khỏi stack
  3. Cho biết stack có rỗng hay không?
  4. Kết thúc chương trình.

# Ví dụ: Bài toán "tính toán liên tiếp" (computing spans)

- Chúng ta chỉ ra làm thế nào sử dụng một stack để tạo ra cấu trúc dữ liệu bổ trợ cho giải thuật
- Cho một mảng X, the span S[i] của X[i] là số lượng lớn nhất các phần tử X[j] ngay sát phía trước X[i] sao cho X[j]≤X[i].
- Spans có các ứng dụng để phân tích tài chính




| X | 6 | 3 | 4 | 5 | 2 |
|---|---|---|---|---|---|
| S | 1 | 1 | 2 | 3 | 1 |

# Thuật toán bậc 2

| Algorithm *spans1*($X$, $n$) | |
|---|---|
| **Input** array $X$ of $n$ integers | |
| **Output** array $S$ of spans of $X$ | # |
| $S \leftarrow$ new array of $n$ integers | $n$ |
| **for** $i \leftarrow 0$ **to** $n-1$ **do** | $n$ |
| $s \leftarrow 1$ | $n$ |
| **while** $s \leq i \wedge X[i-s] \leq X[i]$ | $1+2+\ldots+(n-1)$ |
| $s \leftarrow s+1$ | $1+2+\ldots+(n-1)$ |
| $S[i] \leftarrow s$ | $n$ |
| **return** $S$ | 1 |

Thuật toán span1 chạy trong thời gian $O(n^2)$

# Tính span với một stack

- Chúng ta lưu trữ chỉ số của các phần tử hiện tại để sử dụng khi "quay lại tìm kiếm"
- Chúng ta duyệt mảng từ trái qua phải
  - Đặt i là chỉ số hiện tại
  - Ta pop những chỉ số từ Stack đến khi tìm thấy một chỉ số j mà X[i]<X[j]
  - Ta đặt S[i]←i-j
  - Ta push i vào Stack

# Thuật toán tuyến tính

- Mỗi một chỉ số của mảng thì được:
  - push vào stack chính xác 1 lần
  - pop từ mảng ra nhiều nhất một lần
- Vòng lặp while-loop thực hiện nhiều nhất n lần
- Thuật toán span2 có thời gian chạy là O(n)

```
Algorithm spans2(X, n)                          #
    S ← new array of n integers                 n
    A ← new empty stack                         1
    for i ← 0 to n − 1 do                       n
        while (¬A.isEmpty() ∧                   
                X[top()] ≤ X[i] ) do            n
            j ← A.pop()                         n
        if A.isEmpty() then                     n
            S[i] ← i + 1                        n
        else
            S[i] ← i − j                        n
        A.push(i)                               n
    return S                                    1
```

# Ký pháp Ba Lan

- Trong toán học các biểu thức được viết theo ký pháp trung tố.

  Ví dụ: a*(b+c)-(d*a)
- Nhà toán học Ba Lan Lukasiewicz đưa ra hai dạng ký pháp biểu diễn biểu thức đó là ký pháp tiền tố (prefix) và hậu tố (postfix).
- Ký pháp tiền tố: Các toán tử đứng trước các toán hạng
  - Ví dụ:  a*(b+c)-(d*a) ký pháp tiền tố là -*a+bc*da
- Ký pháp hậu tố: Các toán tử đứng sau các toán hạng
  - Ví dụ:  a*(b+c)-(d*a) ký pháp hậu tố là abc+*da*-

# Ký pháp Ba Lan (tiếp)

- Các dạng ký pháp tiền tố và hậu tố biểu diễn các biểu thức được gọi là ký pháp BaLan và ký pháp BaLan ngược
- Biểu diễn các biểu thức theo ký pháp Ba Lan có một số ưu điểm sau:
  - Không sử dụng các dấu (,)
  - Dễ dàng lập trình để tính giá trị các biểu thức

# Ký pháp Ba Lan (tiếp)

- Thuật toán chuyển biểu thức biểu diễn theo ký pháp trung tố về dạng biểu diễn ký pháp hậu tố
  - Sử dụng 2 Stack Opr (lưu các toán tử trong quá trình chuyển) và BLExp (lưu biểu thức dạng hậu tố)
  - Thuật toán
    - Đọc lần lượt từ trái qua phải biểu thức dạng trung tố
    - Nếu gặp dấu ( thì PUSH nó vào Opr
    - Nếu gặp toán hạng thì PUSH vào BLExp
    - Nếu gặp dấu ) thì POP các toán tử của Opr và PUSH vào BLExp đến khi gặp dấu ( thì POP dấu ( bỏ nó đi.
    - Nếu gặp toán tử thì:
      - Nếu Opr rỗng thì PUSH toán tử đó vào Opr
      - Nếu toán tử được PUSH vào Opr cuối cùng có mức ưu tiên cao hơn toán tử vừa đọc thì: lần lượt POP các toán tử ra khỏi Opr và PUSH vào BLExp, đến khi gặp toán tử có mức ưu tiên thấp hơn hoặc Opr rỗng thì dừng lại. PUSH toán tử vào Opr
      - Nếu toán tử được PUSH vào Opr cuối cùng có mức ưu tiên nhỏ hơn toán tử vừa đọc thì: PUSH toán tử vừa đọc vào Opr
    - Cuối cùng POP tất cả các toán tử còn lại trong Opr và PUSH vào BLExp
    - Đảo ngược thứ tự các phần tử trong BLExp ta được biểu thức dạng hậu tố

      Thứ tự ưu tiên các toán tử (giảm dần): *, /, -, +, ), (

# Ký pháp Ba Lan (tiếp)

**Ví dụ**: Chuyển biểu thức A*B+C*((D-E)+F)/G từ dạng trung tố sang dạng hậu tố

| Token | Stack | Output |
|---|---|---|
| A | {Empty} | A |
| * | * | A |
| B | * | A B |
| + | + | A B * |
| C | + | A B * C |
| * | + * | A B * C |
| ( | + * ( | A B * C |
| ( | + * ( ( | A B * C |
| D | + * ( ( | A B * C D |
| - | + * ( ( - | A B * C D |
| E | + * ( ( - | A B * C D E |
| ) | + * ( | A B * C D E - |
| + | + * ( + | A B * C D E - |
| F | + * ( + | A B * C D E – F |
| ) | + * | A B * C D E – F + |
| / | + / | A B * C D E – F + * |
| G | + / | A B * C D E – F + * G |
| | {Empty} | A B * C D E – F + * G / + |

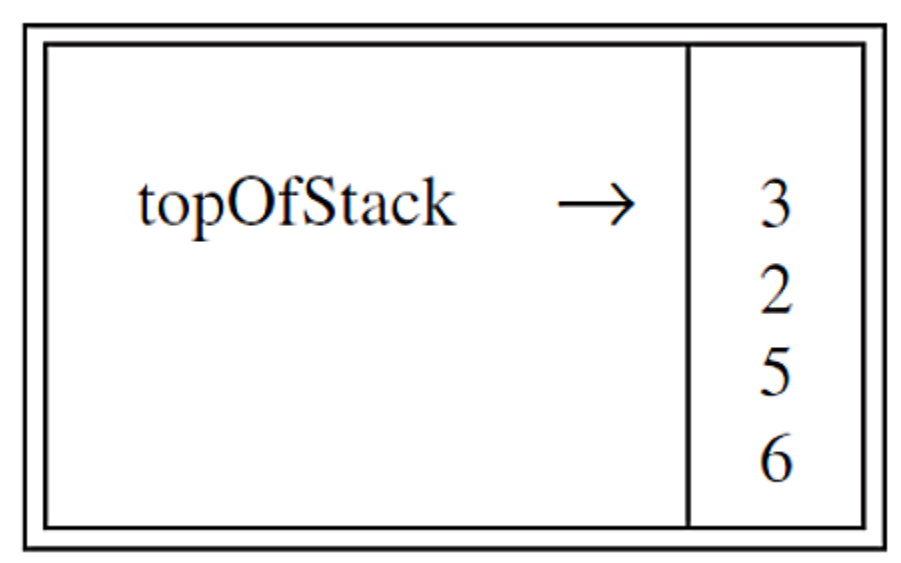
# Ký pháp Ba Lan (tiếp)

- Thuật toán tính giá trị của biểu thức dạng hậu tố
  - Biểu thức lưu trong Stack BLExp, sử dụng stack phụ T
  - Thực hiện POP lần lượt các phần tử trong BLExp
    - Nếu gặp toán hạng thì PUSH nó vào T
    - Nếu gặp toán tử thì:
      - POP 2 phần tử đầu của T ra và thực hiện với toán tử đó, PUSH kết quả thu được vào T.
  - Quá trình thực hiện cho đến khi BLExp rỗng. Giá trị của biểu thức là phần tử còn lại trong T.

# Ký pháp Ba Lan (tiếp)

- Ứng dụng của ngăn xếp
  - Định giá biểu thức hậu tố
  - **Ví dụ**: Định giá biểu thức **6 5 2 3 + 8 ∗ + 3 + ∗**

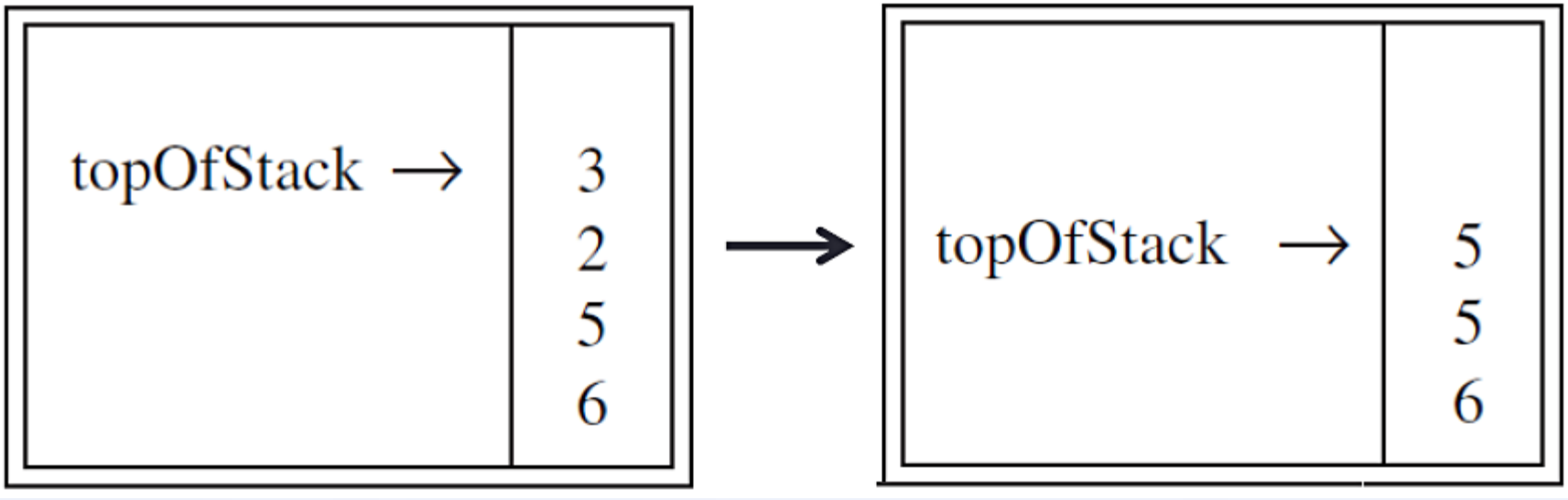
Đặt bốn toán hạng đầu tiên vào ngăn xếp



# Ký pháp Ba Lan (tiếp)

- Ứng dụng của ngăn xếp
  - Định giá biểu thức hậu tố
  - **Ví dụ**: Định giá biểu thức **6 5 2 3 + 8 * + 3 + ***

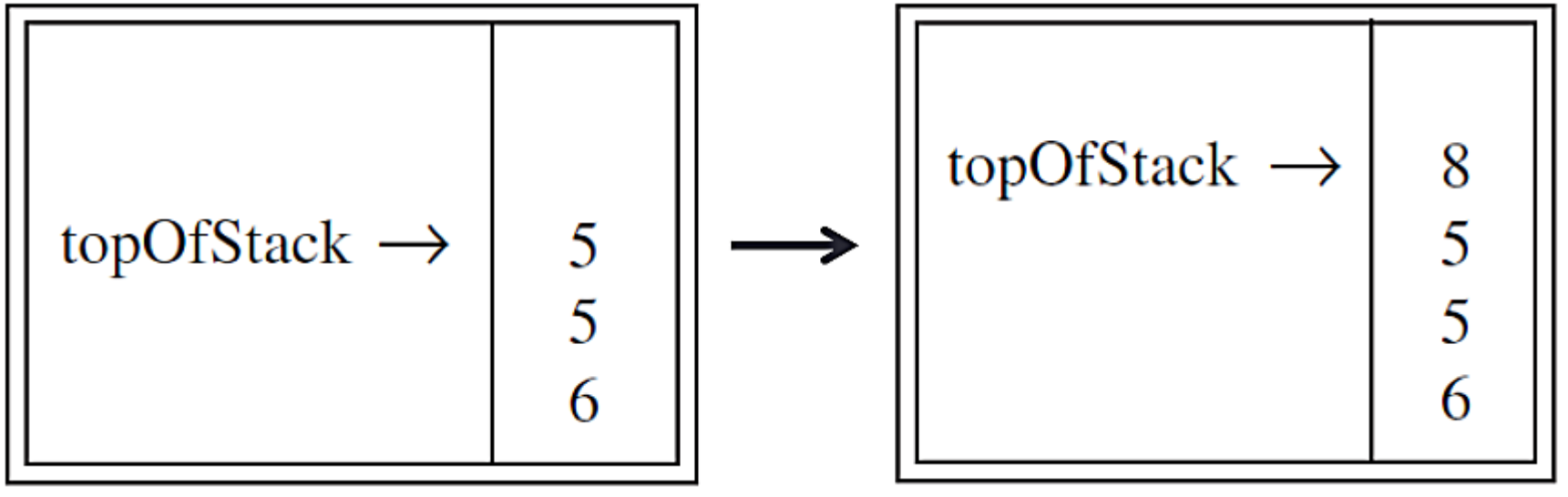
Đọc "+", lấy 3 và 2 ra, cộng lại được 5, đặt 5 vào ngăn xếp



# Ký pháp Ba Lan (tiếp)

- Ứng dụng của ngăn xếp
  - Định giá biểu thức hậu tố
  - **Ví dụ**: Định giá biểu thức **6 5 2 3 + 8 * + 3 + ***

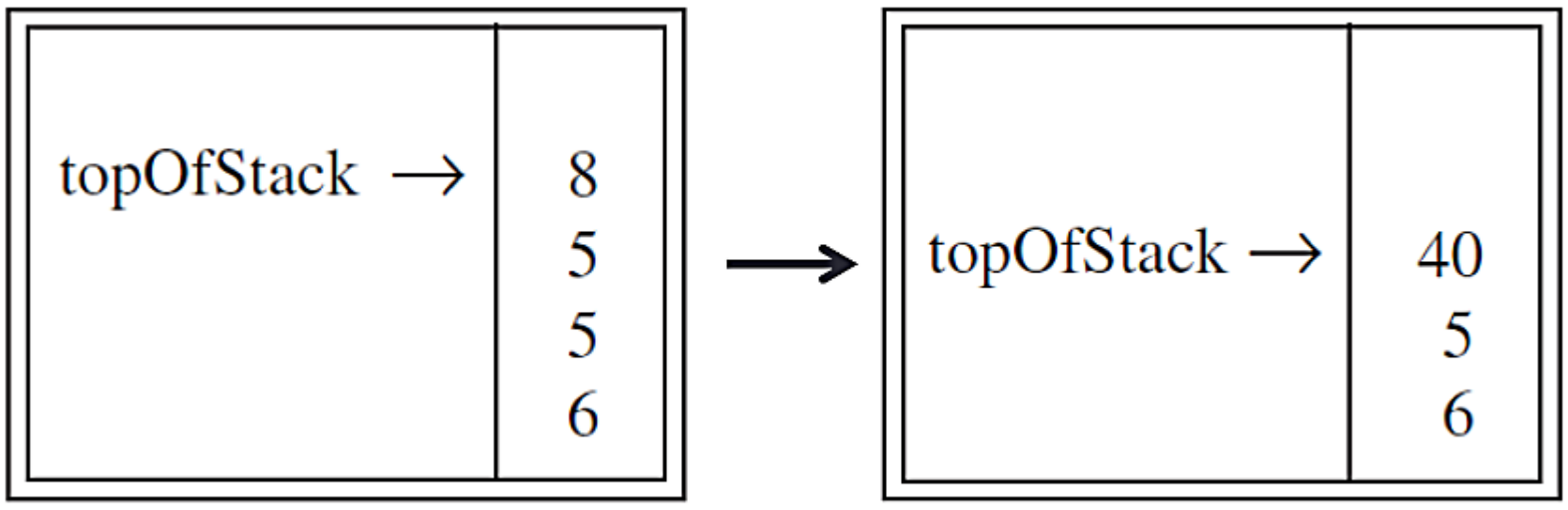
Đặt 8 vào ngăn xếp



# Ký pháp Ba Lan (tiếp)

- Ứng dụng của ngăn xếp
  - Định giá biểu thức hậu tố
  - **Ví dụ**: Định giá biểu thức **6 5 2 3 + 8 * + 3 + ***

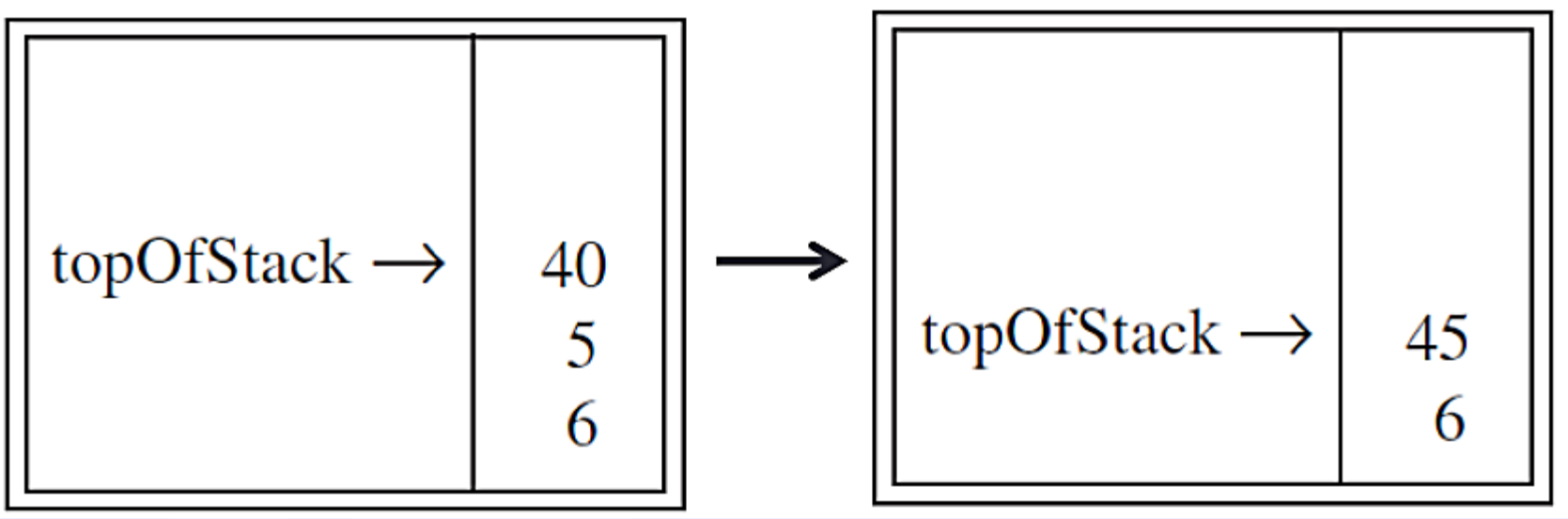
Đặt '*', lấy 8 và 5 ra và nhân với nhau được 40 và đặt 40 vào ngăn xếp



# Ký pháp Ba Lan (tiếp)

- Ứng dụng của ngăn xếp
  - Định giá biểu thức hậu tố
  - **Ví dụ**: Định giá biểu thức **6 5 2 3 + 8 * + 3 + ***

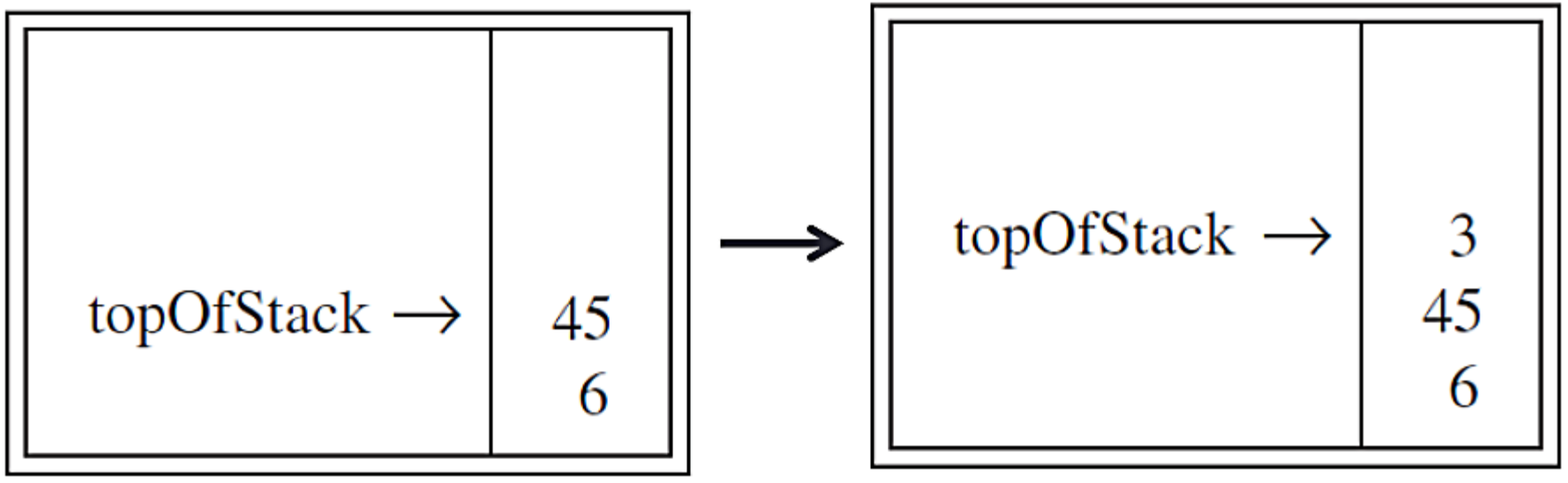
Đọc "+", lấy 40 và 5 ra, cộng lại được 45, đặt 45 vào ngăn xếp



# Ký pháp Ba Lan (tiếp)

- Ứng dụng của ngăn xếp
  - Định giá biểu thức hậu tố
  - **Ví dụ**: Định giá biểu thức **6 5 2 3 + 8 * + 3 + ***

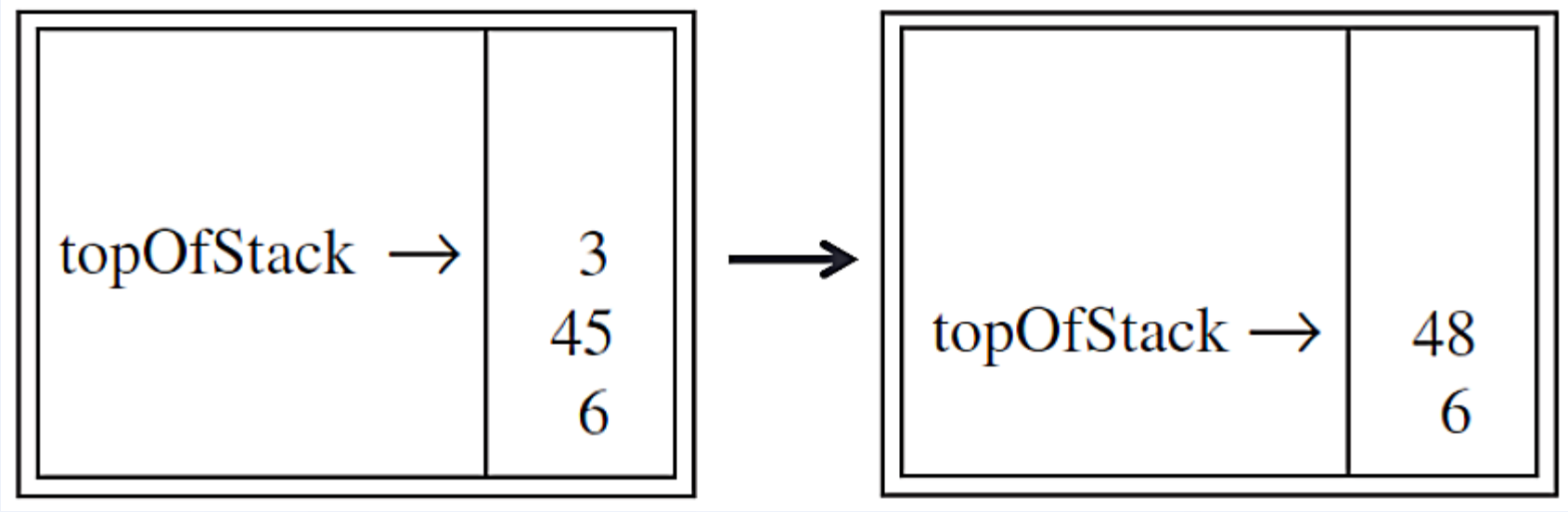
Đặt 3 vào ngăn xếp



# Ký pháp Ba Lan (tiếp)

- Ứng dụng của ngăn xếp
  - Định giá biểu thức hậu tố
  - **Ví dụ**: Định giá biểu thức **6 5 2 3 + 8 * + 3 + ***

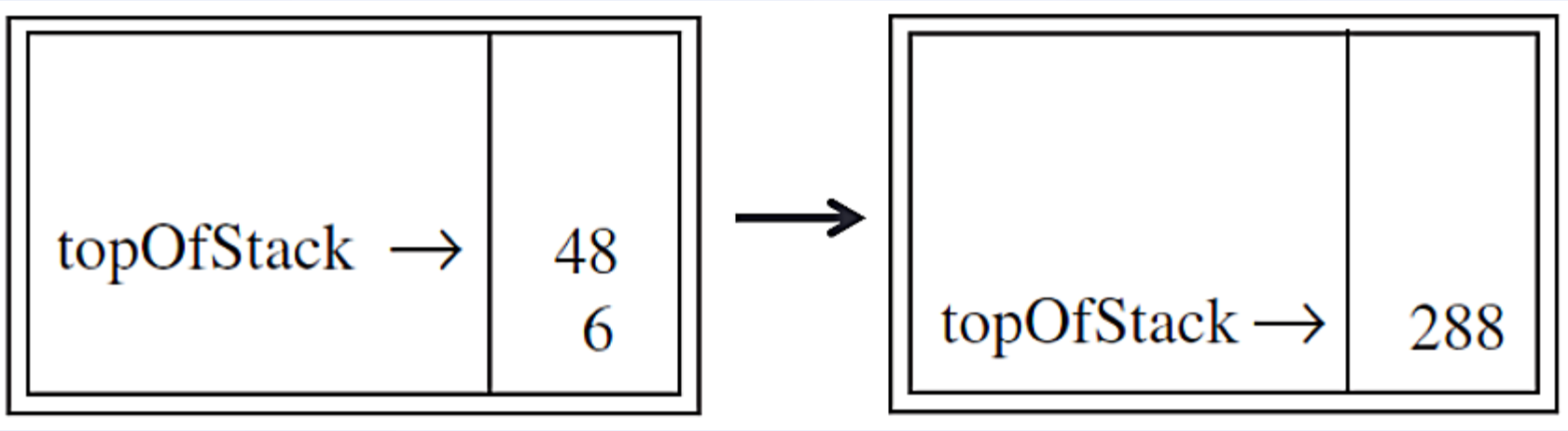
Đọc "+", lấy 3 và 45 ra, cộng lại được 48, đặt 48 vào ngăn xếp



# Ký pháp Ba Lan (tiếp)

- Ứng dụng của ngăn xếp
  - Định giá biểu thức hậu tố
  - **Ví dụ**: Định giá biểu thức **6 5 2 3 + 8 * + 3 + ***

Đọc "*", lấy 48 và 6 ra, nhân vào được 288, đặt 288 vào ngăn xếp



# Cải tiến cài đặt Stack bằng mảng

- Khi thực hiện phép toán push, trong khi mảng đầy sẽ dẫn đến ngoại lệ. Ta có thể thay thế mảng bằng mảng có kích thước lớn hơn
- Làm thế nào để thay thế bằng một mảng lớn hơn?

  - Chiến lược gia tăng:

  Thay thế mảng cũ bằng một mảng mới với kích thước bẳng kích thước mảng cũ cộng với một hằng số c

  - Chiến lược gấp đôi: Thay thế mảng cũ bằng một mảng mới với kích thước gấp đôi kích thước của mảng cũ

# So sánh hai chiến lược

- Chúng ta so sánh chiến lược gia tăng và chiến lược gấp đôi bằng việc phân tích tổng thời gian T(n). T(n) là tổng thời gian cần thiết để hoàn thành phép toán push toàn bộ một chuỗi n phần tử vào Stack
- Chúng ta bắt đầu với một Stack rỗng và thể hiện Stack bằng mảng có kích thước là 1
- Chúng ta gọi thời gian chạy trung bình của phép tóan push chuỗi n phần tử là T(n)/n

# Phân tích chiến lược gia tăng

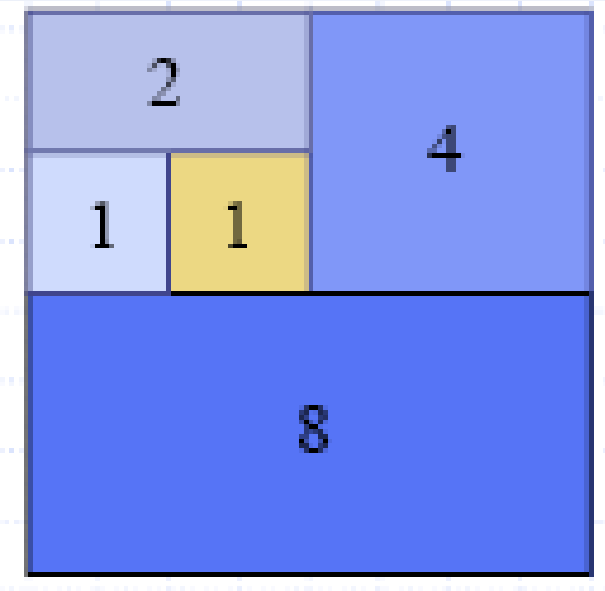
- Chúng ta phải thực hiện thay thế mảng k=n/c lần
- Tổng thời gian T(n) của việc push chuỗi n phần tử vào stack tương ứng với:

  $n+c+2c+3c+\ldots+kc = n+c(1+2+\ldots+k) = n+ck(k+1)/2$
- Khi c là một hằng số thì T(n) là $O(n+k^2)$ và như vậy $T(n) = O(n^2)$
- Thời gian thực hiện phép toán push là O(n)

# Phân tích chiến lược gấp đôi

- Chúng ta thay thế mảng $k=\log_2 n$ lần
- Tổng thời gian T(n) của việc push chuỗi n phần tử vào Stack tương ứng với:

  $n+1+2+4+8+\ldots+2^k = n + 2^{k+1}-1 = 2n-1$
- T(n) là O(n)
- Thời gian thực hiện phép toán push là O(1)



# Cài đặt Stack trong C++

```
template<class Object>
class ArrayStack{
private:
    int capacity; //Số phần tử tối đa
    Object *S;
    int t;  //Lưu chỉ số pt vào cuối cùng
public:
    ArrayStack(int c)
    bool empty();
    int size();
    int top();
    int push(Object o);
    int pop(Object &o);
}
```

```
template<class Object>
ArrayStack<Object>::ArrayStack(int c){
        capacity = c;
        S = new Object[capacity];
        t=-1;
    }
template<class Object>
int ArrayStack<Object>::isEmpty(){
        return(t<0);
    }
template<class Object>
int ArrayStack<Object>::size()
    { return t+1; }
```

# Cài đặt stack trong C++

- Cài đặt stack bằng mảng: https://ideone.com/AuYDoZ
- Cài đặt stack bằng liên kết node đơn
  - https://ideone.com/s6RpaH

Bài tập 1.

Viết chương trình cho phép nhập vào một biểu thức dạng trung tố bất kỳ. Tính giá trị của biểu thức đó.

Đề bài:

http://laptrinhonline.club/problem/tichpxhtbalan

Code tham khảo

https://ideone.com/tSMnPL

Bài tập 2.

Viết chương trình tính khối lượng hóa chất

Đề bài:

http://laptrinhonline.club/problem/tichpxweighthoachat

Code tham khảo

https://ideone.com/rr3OVC

## Bài tập 3. Thuật toán DFS cho bài toán mọi con đường về không

Đề bài:

http://laptrinhonline.club/problem/tichpxlietkezero

Code tham khảo

https://ideone.com/qccSNd

# Bài tập

**Bài 1. Cài đặt lớp Stack bằng danh sách liên kết.**

**Bài 2. Xây dựng lớp ứng dụng Stack quản lý các phần tử là các số nguyên. Có các chức năng sau:**

1. Thêm một phần tử vào stack
2. Lấy ra một phần tử của Stack.
3. Cho biết Stack rỗng hay không
4. Cho biết Stack có bao nhiêu phần tử.