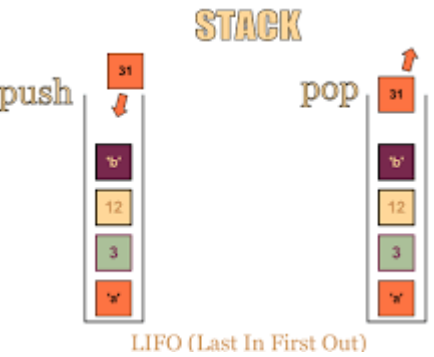
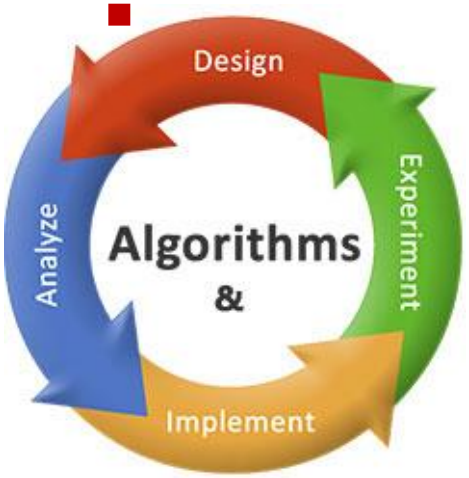
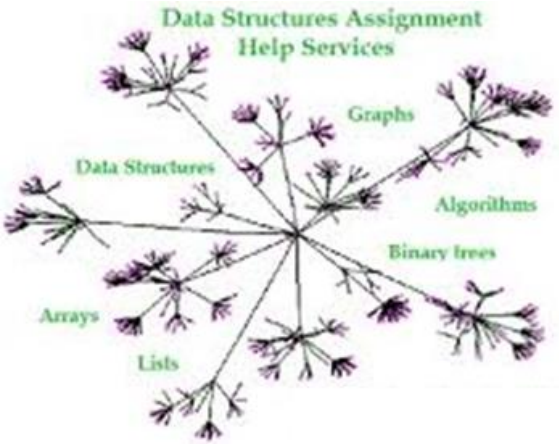


CẤU TRÚC DỮ LIỆU

&

GIẢI THUẬT





Lê Văn Hạnh

levanhanhvn@gmail.com

NỘI DUNG MÔN HỌC

- Chương 1: Ôn tập ngôn ngữ lập trình C
- Chương 2: Kiểu dữ liệu con trỏ
- Chương 3: Tổng quan về cấu trúc dữ liệu và giải thuật
- Chương 4: Danh sách kề (Danh sách tuyến tính)
- Chương 5: Các giải thuật tìm kiếm trên danh sách kề
- Chương 6: Các giải thuật sắp xếp trên danh sách kề
- Chương 7: Danh sách liên kết động *(Linked List)*
- **Chương 8: Ngăn xếp** *(Stack)*
- Chương 9: Hàng đợi *(Queue)*
- Chương 10: Cây nhị phân tìm kiếm *(Binary Search Tree)*
- Chương 11: Cây cân bằng *(Balanced binary search tree – AVL tree)*
- Chương 12: Bảng băm *(Hash Table)*

Chương 8

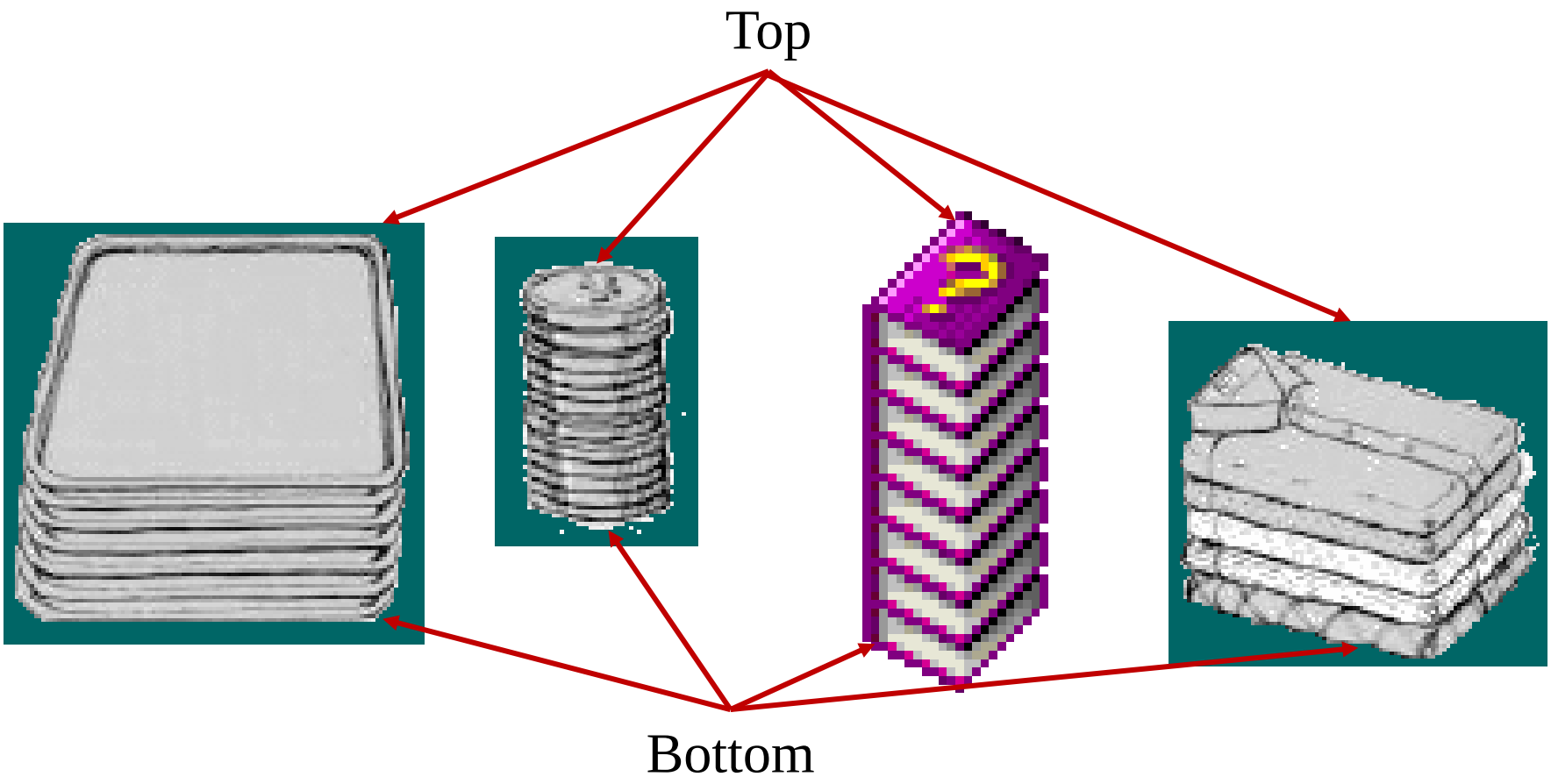
TỔ CHỨC NGĂN XẾP

(Stack)

NỘI DUNG

1. Giới thiệu
2. Định nghĩa
3. Một số ứng dụng của stacks trên máy tính
4. Các thao tác cơ bản
5. Xây dựng Stack dựa trên cấu trúc mảng 1 chiều
6. Xây dựng Stack dựa trên cấu trúc DSLK

1. Giới thiệu



Chồng khay cà phê

Chồng tiền xu

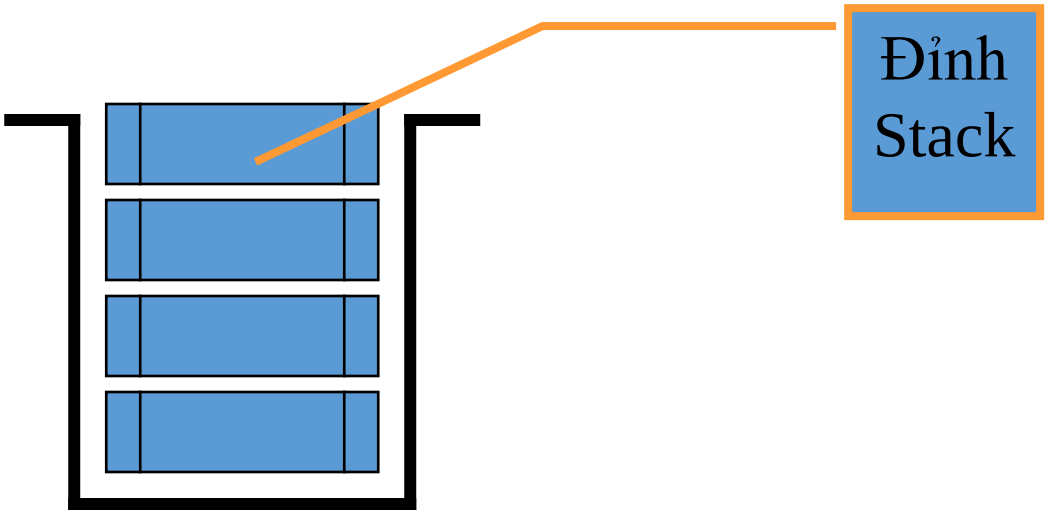
Chồng sách

Chồng áo sơ mi

2. Định nghĩa

- Stack là 1 cấu trúc:
  - Gồm nhiều phần tử có thứ tự.
  - Việc thêm vào hoặc lấy phần tử ra chỉ thực hiện từ 1 phía

⇒ Hoạt động theo cơ chế "***Vào sau – Ra trước***" (LIFO – Last In, First Out)



3. MỘT SỐ ỨNG DỤNG CỦA STACKS TRÊN MÁY TÍNH

Stack thích hợp lưu trữ các loại dữ liệu mà trình tự truy xuất ngược với trình tự lưu trữ

3.1.Compilers

– Kiểm tra sự sắp xếp của các cấu trúc lồng nhau. Vd phân tích cú pháp để kiểm tra tính hợp lệ của các cặp ngoặc (*brackets*) trong 1 biểu thức.

Valid inputs	Invalid Inputs
{}	{(}
({[]})	([(()])
{[]()}	{}[])
[{({}[]({}))}]	[{)}(]}]

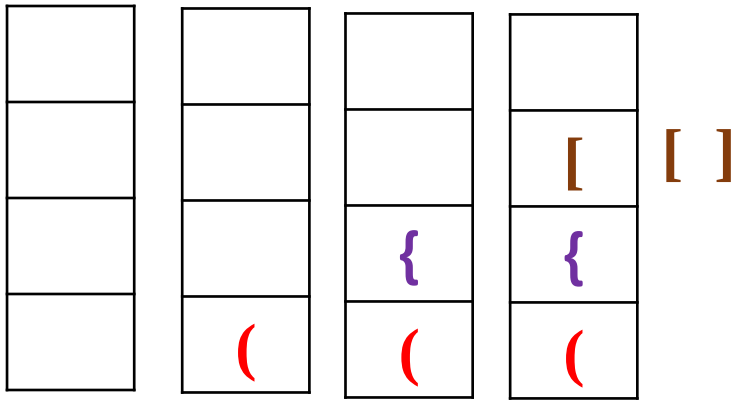
3.1.Compilers

– Minh họa: ({ [ ] })

({ [ ] })

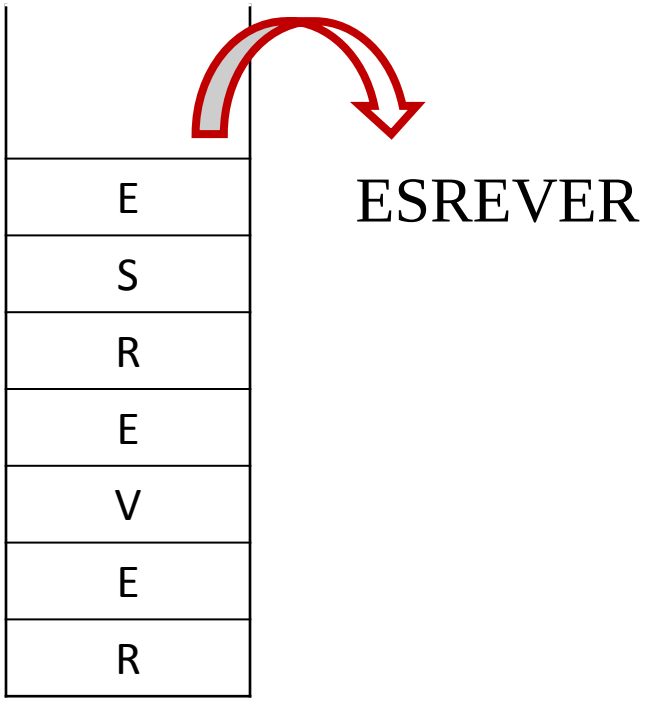
{ }

( )

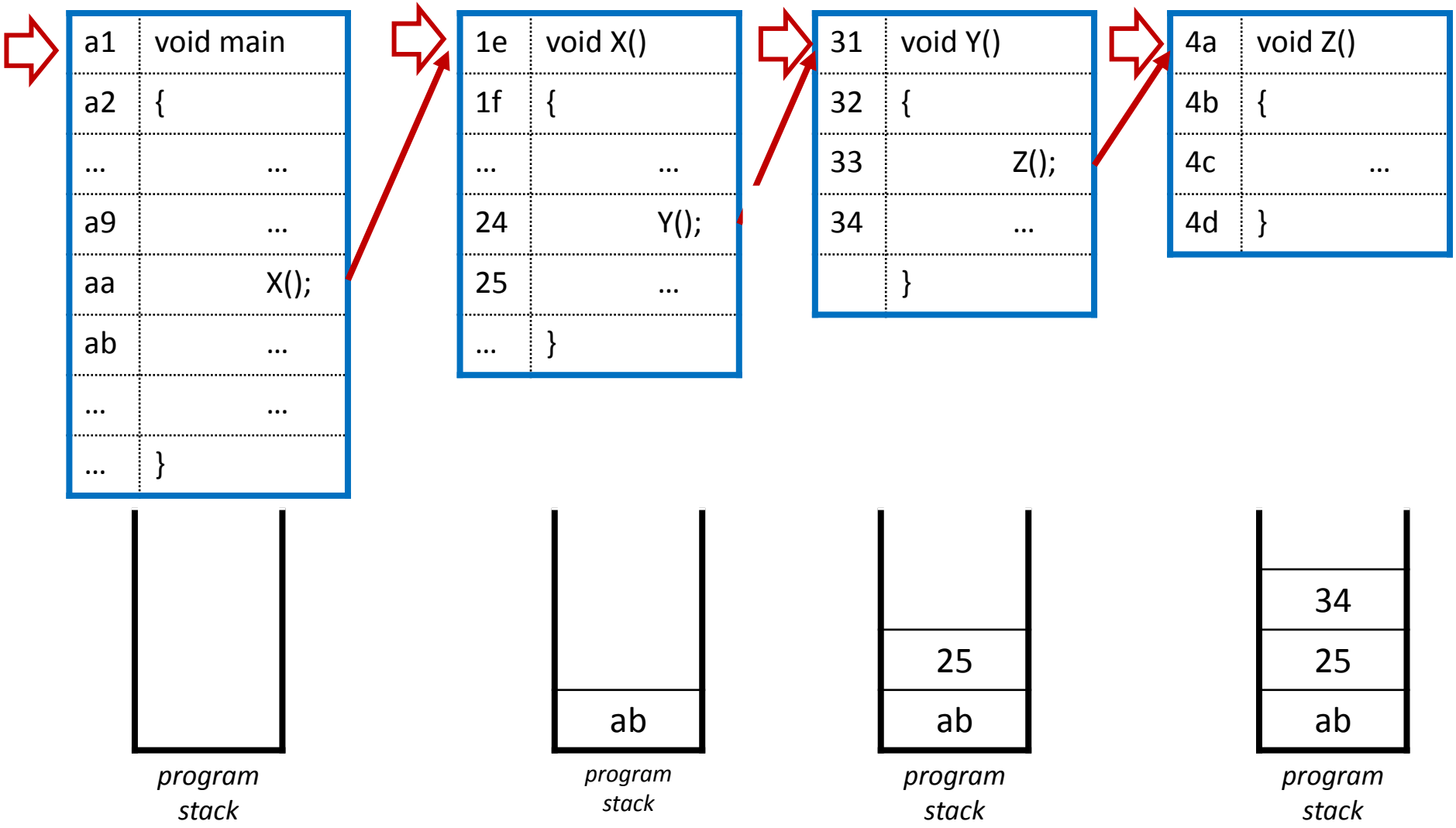


3.2.Xử lý chuỗi: Đảo ngược chuỗi

- Để đảo ngược chuỗi, lần lượt đi từ trái sang phải, cắt từng ký tự đưa vào stack cho đến khi cắt hết chuỗi.
- Lấy từ stack ra để có chuỗi đảo ngược.
- VD: Đảo ngược chuỗi "REVERSE"



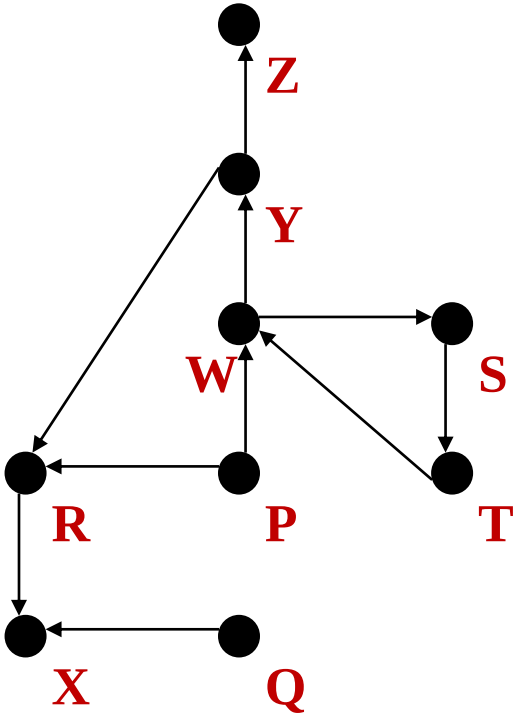
3.3. Operating systems: Thực thi các hàm hoặc phương thức được gọi thông qua *program stack*.



3.4. Artificial intelligence: finding a path

- Theo dõi các lựa chọn trước đó. Vd như trong thuật toán backtracking
- Theo dõi các lựa chọn chưa được thực hiện. Vd như trong việc tạo tìm đường đi trong một mê cung.

3.4. Artificial intelligence: Cho sơ đồ các tuyến đường bay giữa các thành phố.

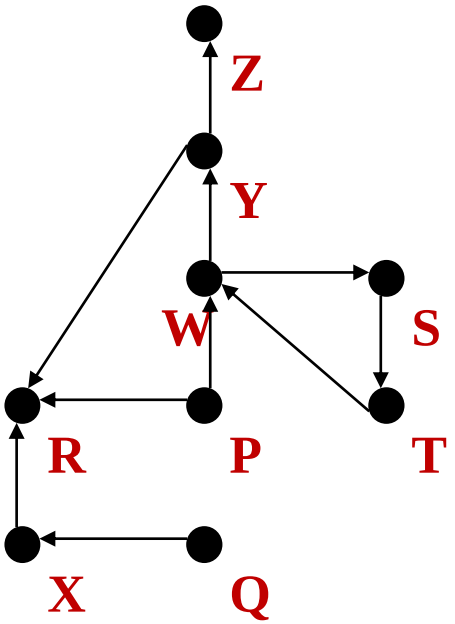


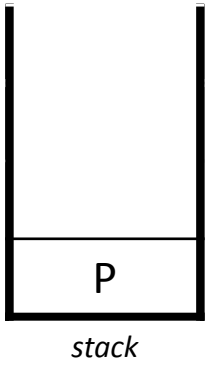
Finding a Path:

Có thể tìm 1 đường đi từ bất kỳ thành phố (TP) ***C1*** nào đến 1 TP ***C2*** khác bằng cách dùng stack:

- Đầu tiên đặt TP khởi hành vào đáy stack và đánh dấu để biết TP đó đã được xét.
- Chọn bất kỳ mũi tên nào ra khỏi TP để đến TP khác với điều kiện TP đến đó chưa được đánh dấu đã được xét.
- Đưa TP mới chọn vào stack.
- Nếu TP mới chọn là TP cần đến thì dừng lại.
- Ngược lại, lặp lại việc chọn tiếp TP khác như cách đã thực hiện ở trên cho đến khi tìm được TP cần đến
- Khi tất cả các TP đều đã được xét nhưng vẫn chưa tìm thấy đường đi ⇒ không có đường đi.

3.4. Artificial intelligence: Cho sơ đồ các tuyến đường bay giữa các thành phố.





Finding a Path:

Cần tìm đường đi từ ***P*** đến ***Y***:

- Đầu tiên đặt ***P*** đáy stack và đánh dấu để biết TP đó đã được xét.
- Chọn (ngẫu nhiên trong số những TP có thể chọn) ***R*** là TP kế tiếp, đưa vào stack và đánh dấu ***R*** đã được xét.
- Từ ***R*** không có đường đi tiếp ⇒ lấy ***R*** ra khoải stack.
- Chọn ***W*** (lựa chọn duy nhất còn lại) là TP kế tiếp, đưa vào stack và đánh dấu ***W*** đã được xét.
- Chọn ***Y*** (ngẫu nhiên trong số những TP có thể chọn), đưa vào stack và đánh dấu ***Y*** đã được xét.
- Do ***Y*** là TP cần đến nên dừng thuật toán.
- Lần lượt lấy ra từ stack để có đường đi: **Y ← W ← P**

3.5. Đổi 1 số từ cơ số 10 sang các số khác

Ví dụ: đổi 57_{10} sang cơ số 2

57

$$57_{10} = 111001_2$$

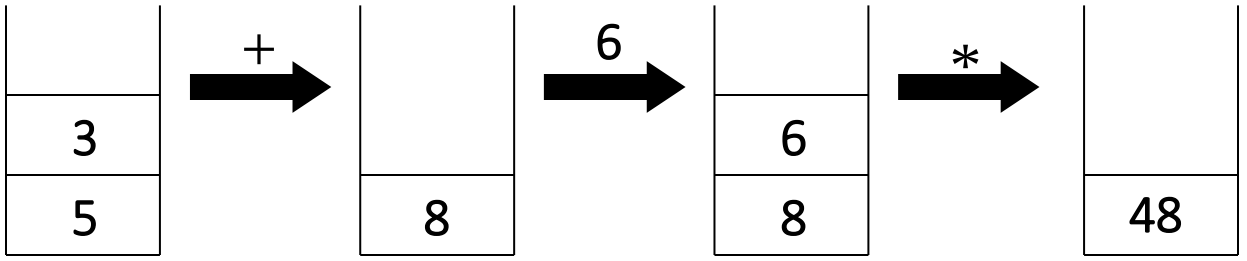
3.6. Virtual machines

- Biểu diễn biểu thức đại số: gồm 3 dạng
  - ***Trung tố*** (*Infix*): đặt toán tử nằm giữa các toán hạng. VD: **a + b**
  - ***Tiền tố*** (*Prefix* hay "*ký pháp Ba Lan*" - *Polish notation*): đặt toán tử lên trước các toán hạng. VD: **+ a b**
  - ***Hậu tố*** (*Postfix* hoặc - "*ký pháp nghịch đảo Ba Lan*" - *Reverse Polish notation*): các toán tử sẽ được đặt sau các toán hạng. VD: **a b +**

3.6. Virtual machines

- Trong máy tính, các phép tính được sắp xếp theo trật tự đã được quy ước trước (*Reverse Polish Notation*). Vd:
  - (5 + 3) * 6 được chuyển thành 5 3 + 6 * (i)
  - 5 + 3 * 6 được chuyển thành 5 3 6 * + (ii)
  - 7–((4+(5*3))/2) được chuyển thành 7 4 5 3 * + 2 / - (iii)
- Tính giá trị của biểu thức dạng hậu tố theo quy ước:
  - Giá trị được đưa vào stack.
  - Duyệt đến phép tính (cộng, trừ, nhân, chia, …), lấy 2 số ra khỏi stack để thực hiện phép tính giữa số lấy ra sau với số lấy ra trước

(i) Tính 5 3 + 6 * ?



(ii) Tính 5 3 6 * + ?

(iii) Tính 7 4 5 3 * + 2 / -

3.6. Virtual machines

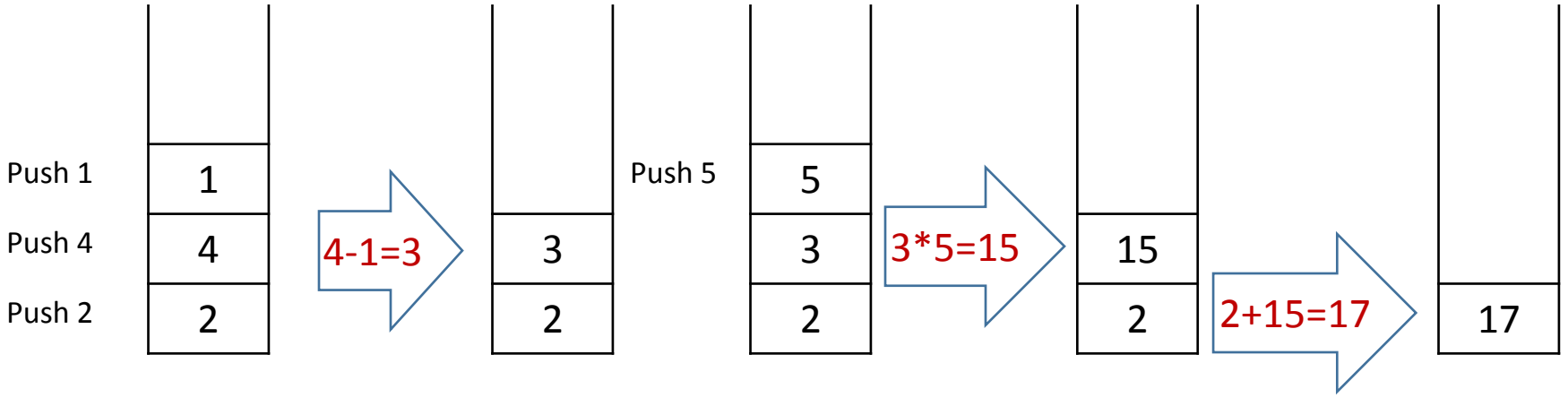
- Chuyển từ Infix sang Postfix:
  - VD1: Cho 2 +(4-1)*5

Step	Expression
1	2 +(4-1)*5
2	2 +41-*5
3	2 +41-5*
4	2 41-5*+

Current Symbol	Action Performed	Stack Status	Postfix Expression
2	*Append to postfix*		2
+	PUSH +	+	2
(	PUSH (	+(	2
4	*Append to postfix*		24
-	PUSH -	+(-	24
1	*Append to postfix*		241
)	POP -	+(	241-
	POP (	+	241-
*	PUSH *	+*	241-
5	*Append to postfix*	+*	241-5
	POP *	+	241-5*
	POP +		241-5*+

3.6. Virtual machines

- Lượng giá các biểu thức dạng Postfix. Vd:
  - Khi duyệt đến phép tính (cộng, trừ, nhân, chia, …), lấy 2 số ra khỏi stack để thực hiện phép tính giữa số lấy ra sau với số lấy ra trước
  - Đặt kết quả vừa tính được vào stack để chờ đến lượt tính toán sau đó.
  - VD: Cho 241-5*+



3. Một số ứng dụng của stacks trên máy tính

3.6. Virtual machines

- Chuyển từ Infix sang Postfix:
  - VD2: Cho

a – (b + c * d) / e

Current Symbol	Action Performed	Stack Status	Postfix Expression
a	*Append to postfix*		a
-	PUSH -	-	a
(	PUSH (	-(	a
b	*Append to postfix*	-(	ab
+	PUSH +	-(+	
c	*Append to postfix*	-(+	abc
*	PUSH *	-(+*	
d	*Append to postfix*	-(+*	abcd
)	POP * - *POP UNTIL (*	-(+	abcd*
	POP +	-(	abcd*+
	POP (	-	abcd*+
/	PUSH /	-/	abcd*+
e	*Append to postfix*		abcd*+e
	POP / - *UNTIL EMPTY*	-	abcd*+e/
	POP -		abcd*+e/-

3.6. Virtual machines

i. Thực hành chuyển từ Infix sang Postfix:

(((a + b) * (c - d) / e) + (m * (n / 2))) * (p / q)

a b + c d – e / * m n 2 / * + p q / *

ii. Cho biết khi chuyển sang hậu tố, những trường hợp nào sau đây có kết quả giống nhau

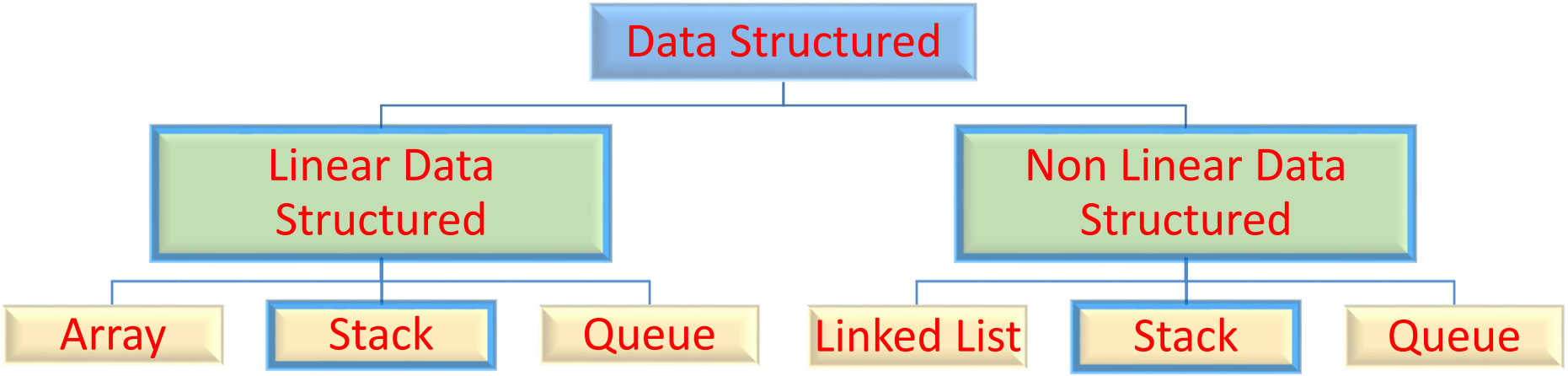
a) (a + b * c) / d

b) (a + (b * c)) / d

c) ((a + b) * c) / d

d) (a * b + c) / d

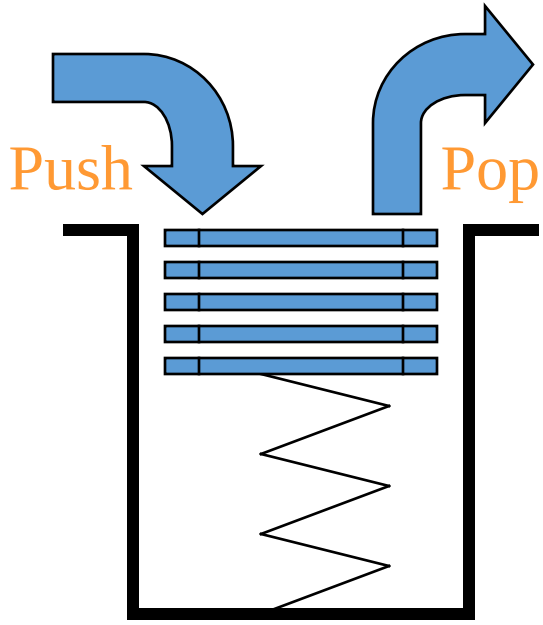
4. CÀI ĐẶT STACK



Có thể chọn 1 trong 2 cách cài đặt stack:

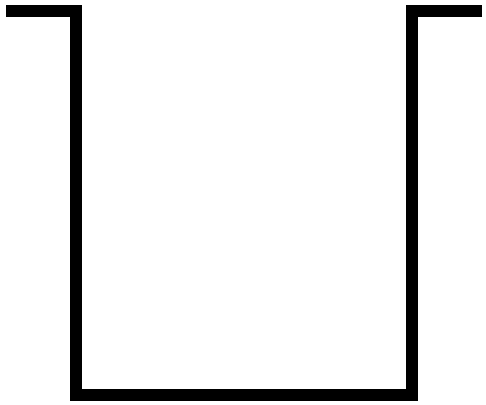
- Sử dụng cấu trúc dữ liệu dạng tuyến tính (cấu trúc dữ liệu tĩnh – danh sách kề mảng 1 chiều).
- Sử dụng cấu trúc dữ liệu dạng phi tuyến (cấu trúc dữ liệu động – danh sách liên kết).

Các thao tác cơ bản trên Stack



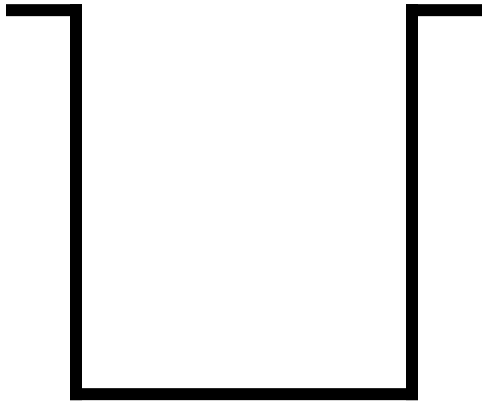
- *InitStack*: khởi tạo Stack rỗng
- *IsEmpty*: kiểm tra Stack rỗng?
- *IsFull*: kiểm tra Stack đầy?
- *Push*: thêm 1 phần tử vào đỉnh Stack
  ⇨ có thể làm Stack đầy
- *Pop*: lấy ra 1 phần tử từ đỉnh Stack
  ⇨ có thể làm Stack rỗng
- *StackTop*: kiểm tra phần tử ở đỉnh Stack
  ⇨ không làm thay đổi Stack

4.1. Minh họa thao tác *InitStack*

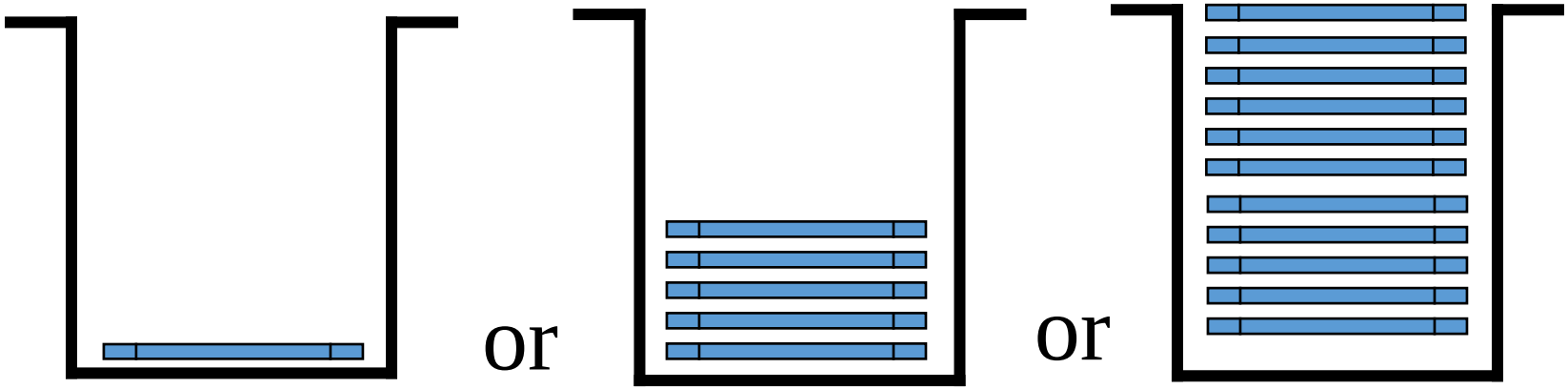


4.2. Minh họa thao tác *IsEmpty*

⇨ return true

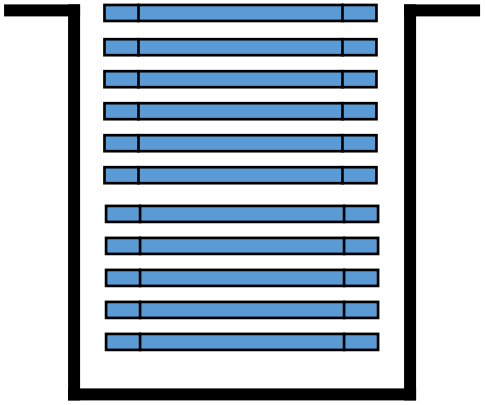


⇨ return false

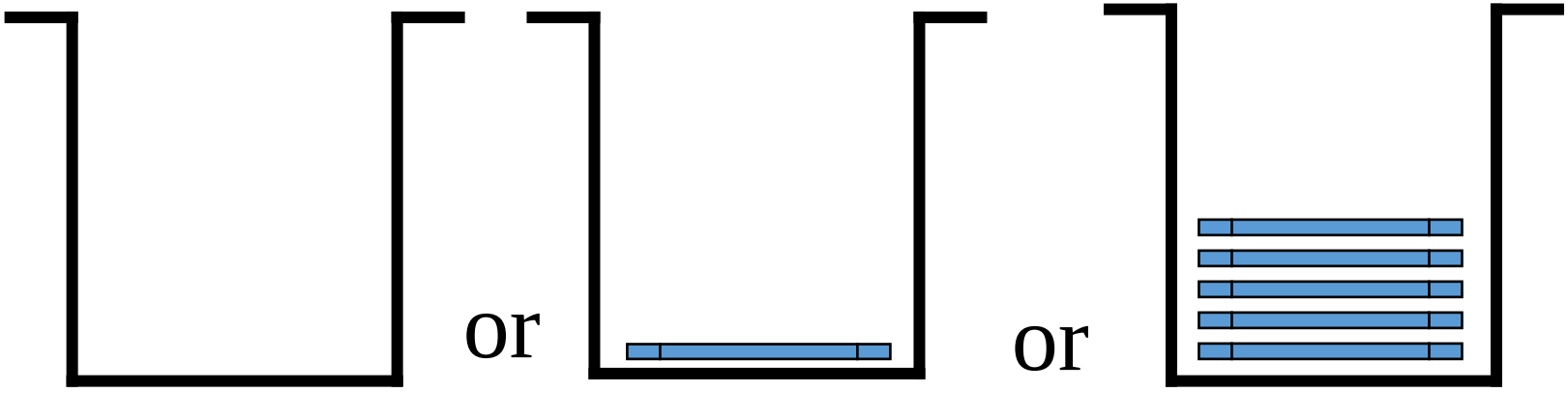


4.3. Minh họa thao tác *IsFull*

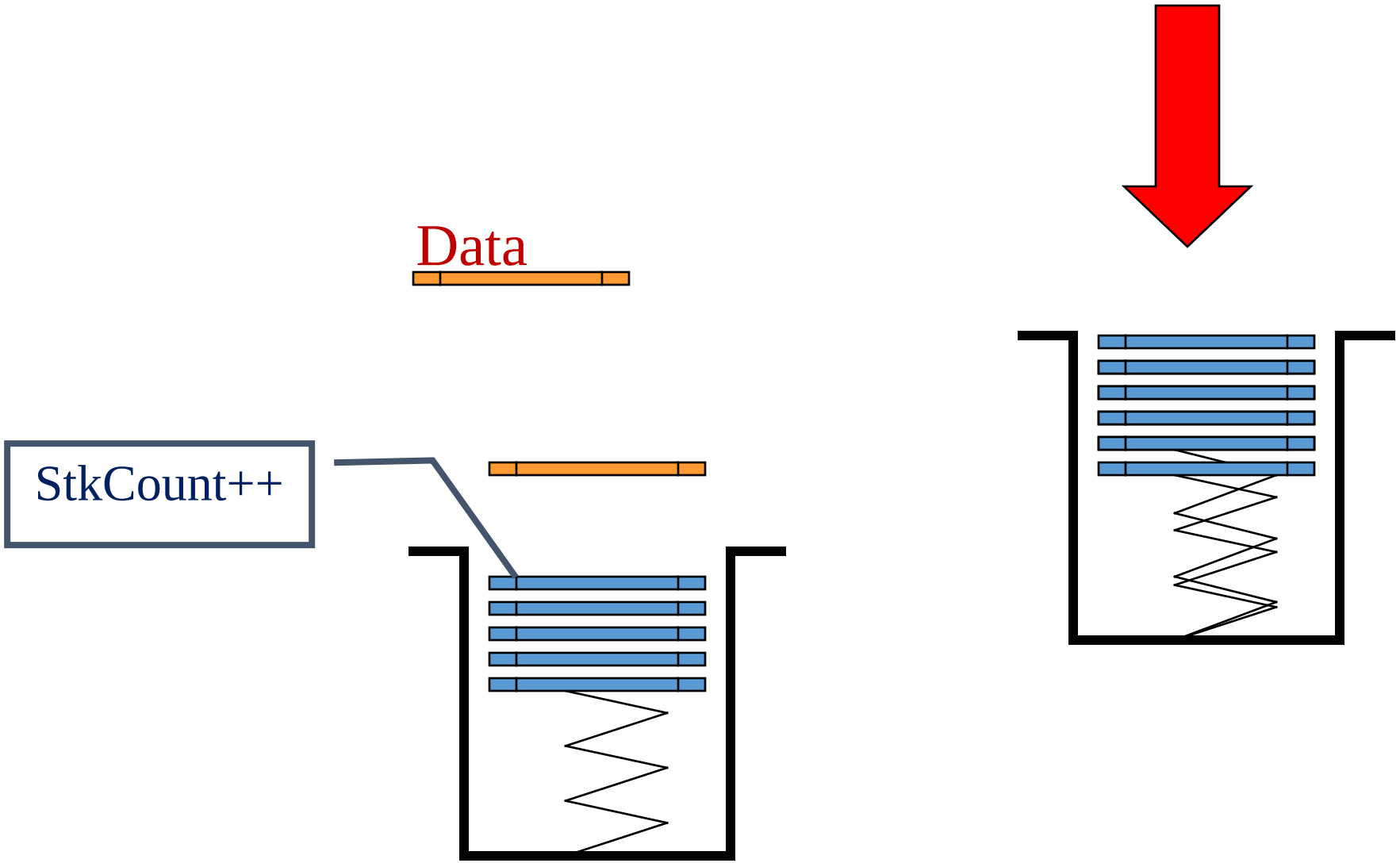
⇨ return true



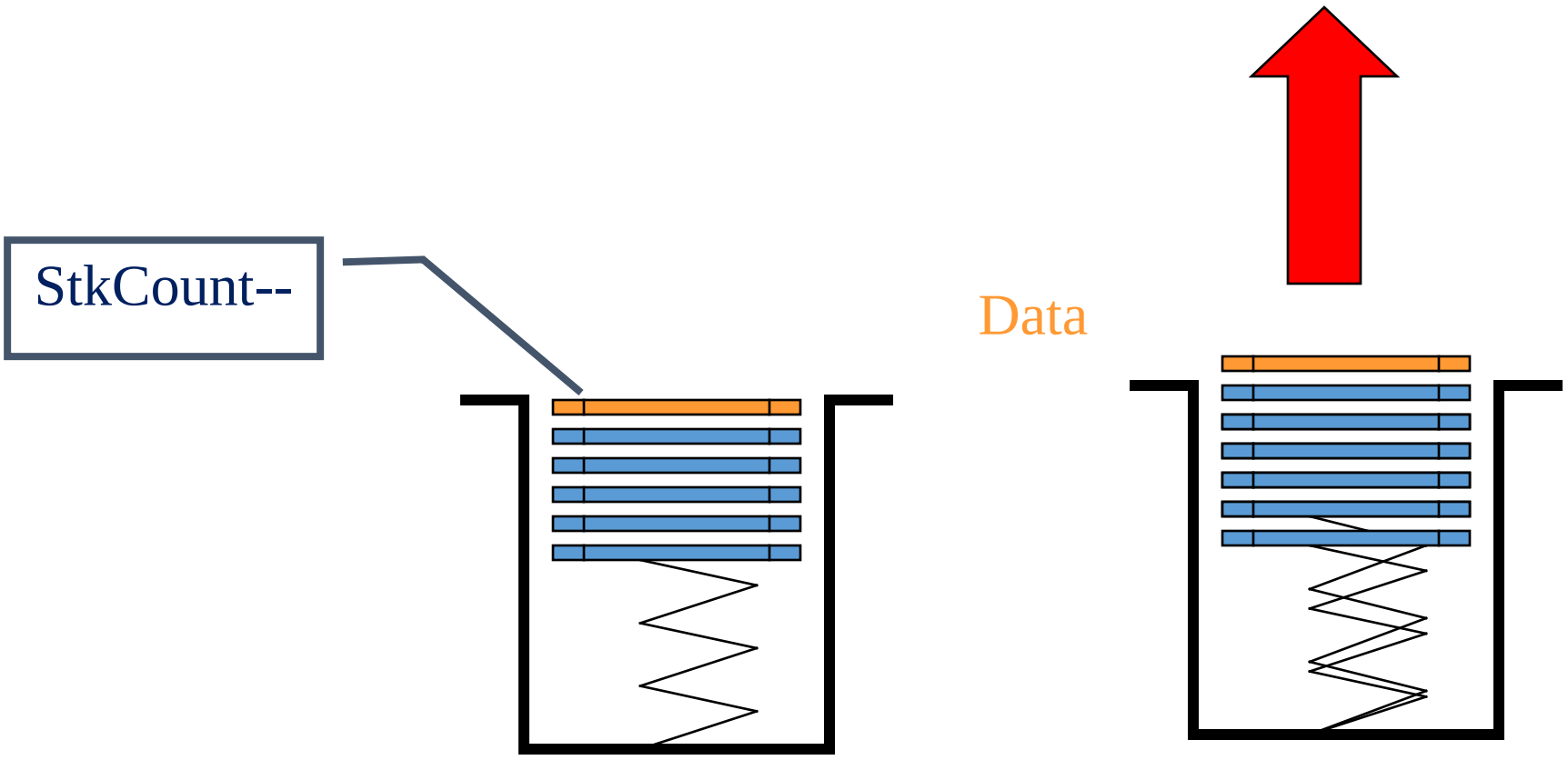
⇨ return false



4.4. Minh họa thao tác *Push*

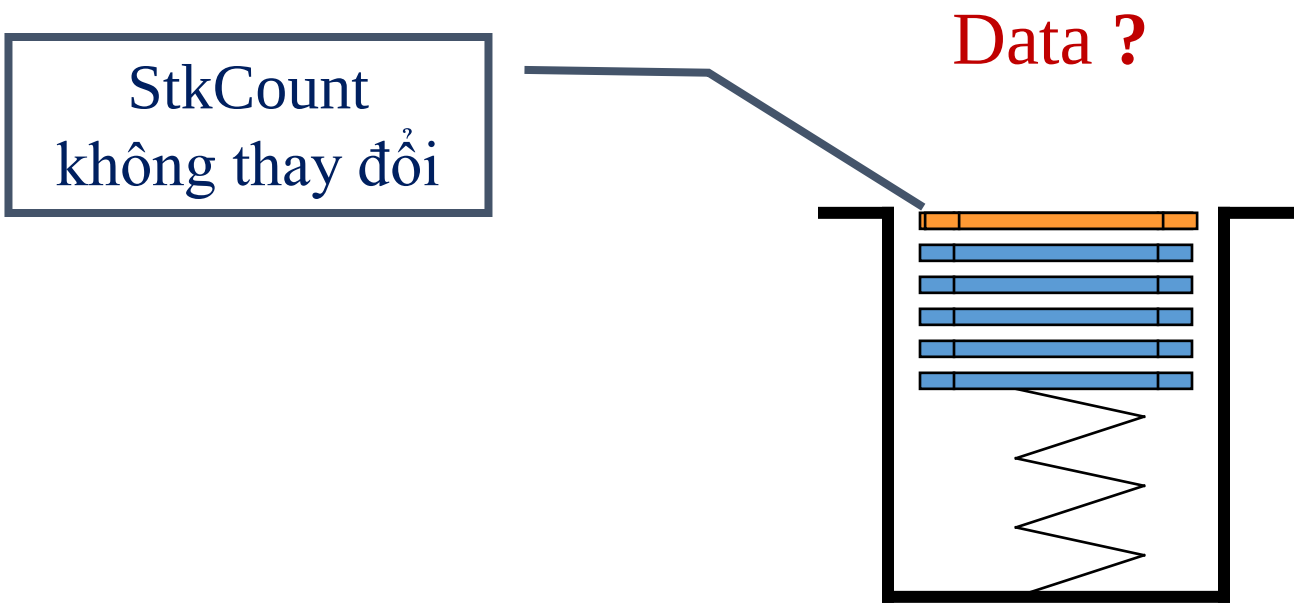


4.5. Minh họa thao tác Pop



4.6. Minh họa thao tác StackTop

⇨ Ngăn xếp không thay đổi

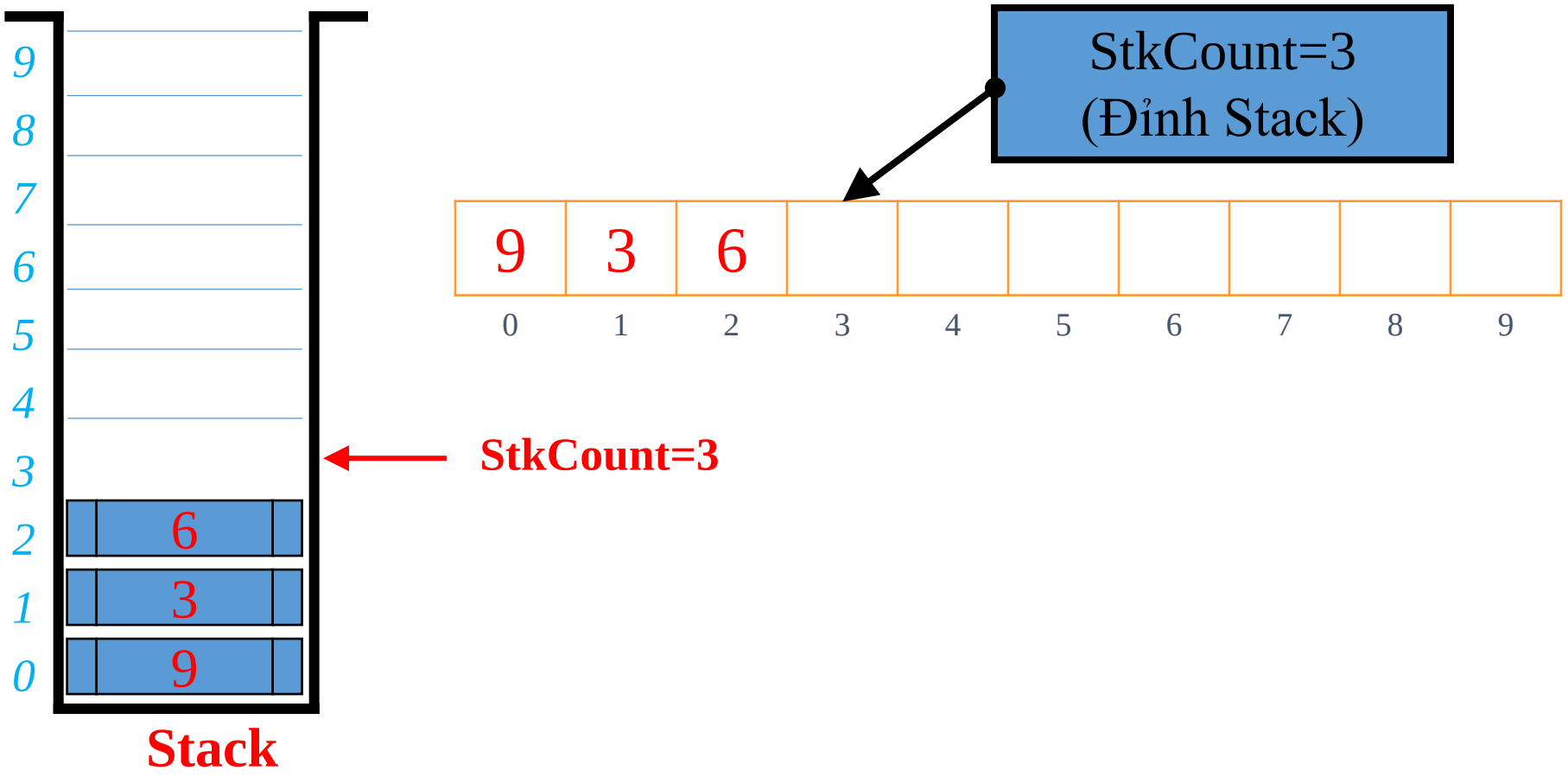


5. SỬ DỤNG DANH SÁCH KỀ LÀM Stack

5.1. Đặc điểm

- Viết chương trình dễ dàng, nhanh chóng.
- Bị hạn chế do số lượng phần tử cố định.
- Tốn chi phí tái cấp phát và sao chép vùng nhớ nếu sử dụng mảng động.

5.2. Minh họa



Minh họa Stack có khả năng chứa tối đa 10 phần tử và hiện đang chứa 3 phần tử

5.3. Cài đặt

5.3.1. Khai báo cấu trúc Stack

//Giả sử Stack chứa các phần tử kiểu nguyên

```
struct STACK
{   int* StkArray;   //mảng chứa các phần tử
    int StkMax;      //số phần tử (sức chứa) tối đa
    int StkCount;    // vị trí đỉnh Stack
};
```

Thực hành: Khai báo cấu trúc stack để chứa các phần tử là:

i. Phân số (gồm tử và mẫu số cùng có kiểu int)

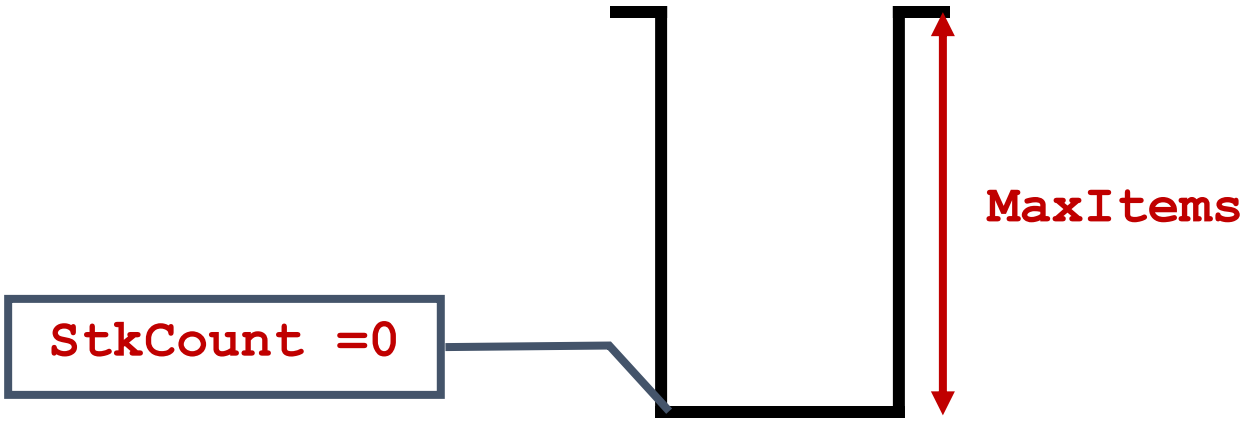
ii. Sinh viên (gồm mã số (int), họ tên (char*), phái (bool), điểm (float))

iii. Hàng hóa (gồm mã số (char*), tên hàng (char*), ngày hết hạn (ngày, tháng, năm kiểu int), đơn giá (float), số lượng tồn (int))

5.3. Cài đặt

5.3.2. Thao tác "Khởi tạo Stack rỗng"

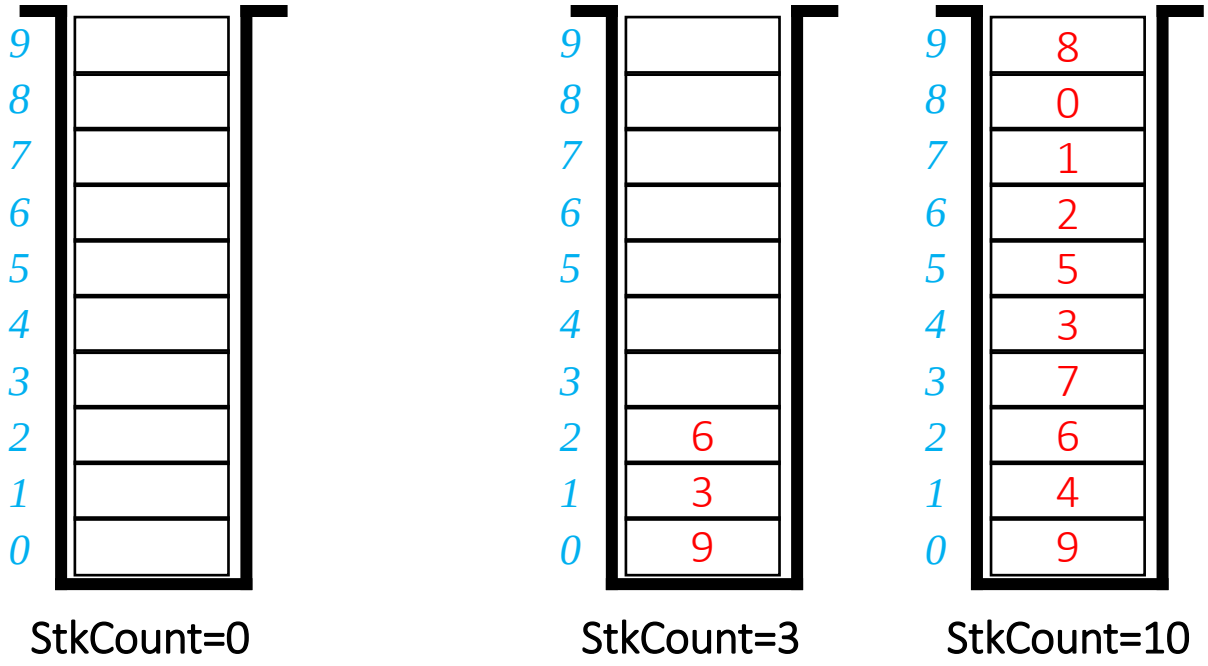
```
int InitStack (STACK &s, int MaxItems)
{
    s.StkArray = new int[MaxItems];
    if (s.StkArray == NULL)
        return 0;   //Không cấp phát được bộ nhớ
    s.StkMax = MaxItems;//sức chứa tối đa
    s.StkCount = 0;//chưa có phần tử nào trong Stack
    return 1;       // khởi tạo thành công
}
```



5.3. Cài đặt

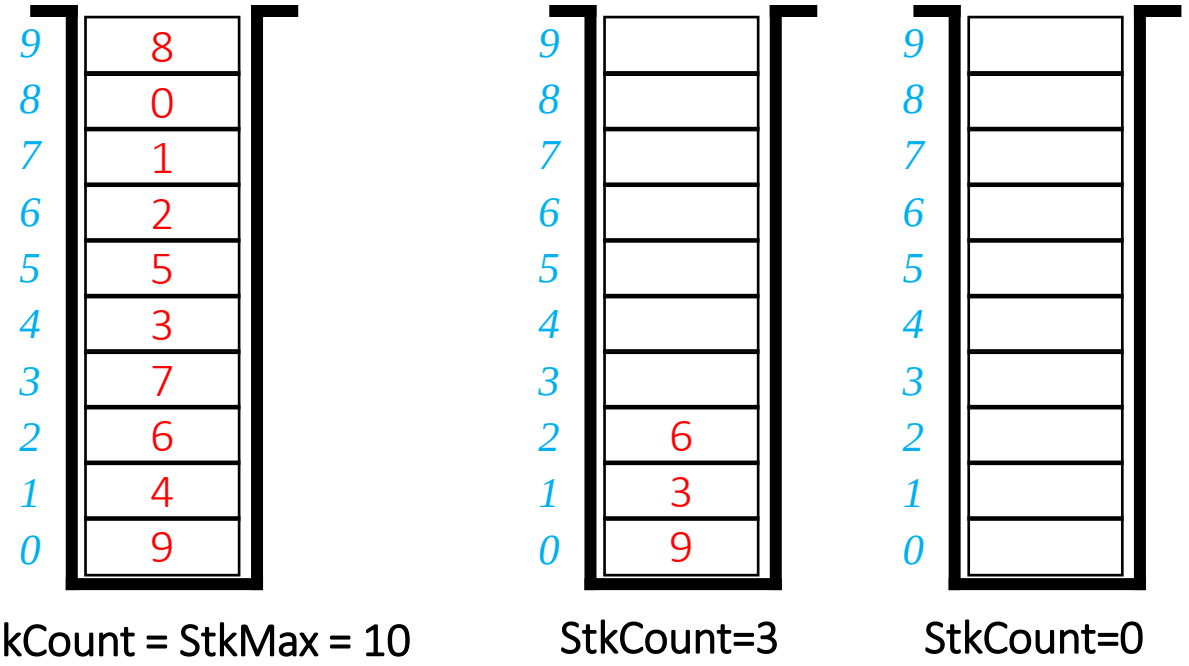
5.3.3. Thao tác "Kiểm tra Stack rỗng"

```
bool IsEmpty(STACK s)
{  /*return true nếu Stack rỗng, return false
                                    nếu không rỗng*/
   return (s.StkCount==0);
}
```



5.3.4. Thao tác "Kiểm tra Stack đầy"

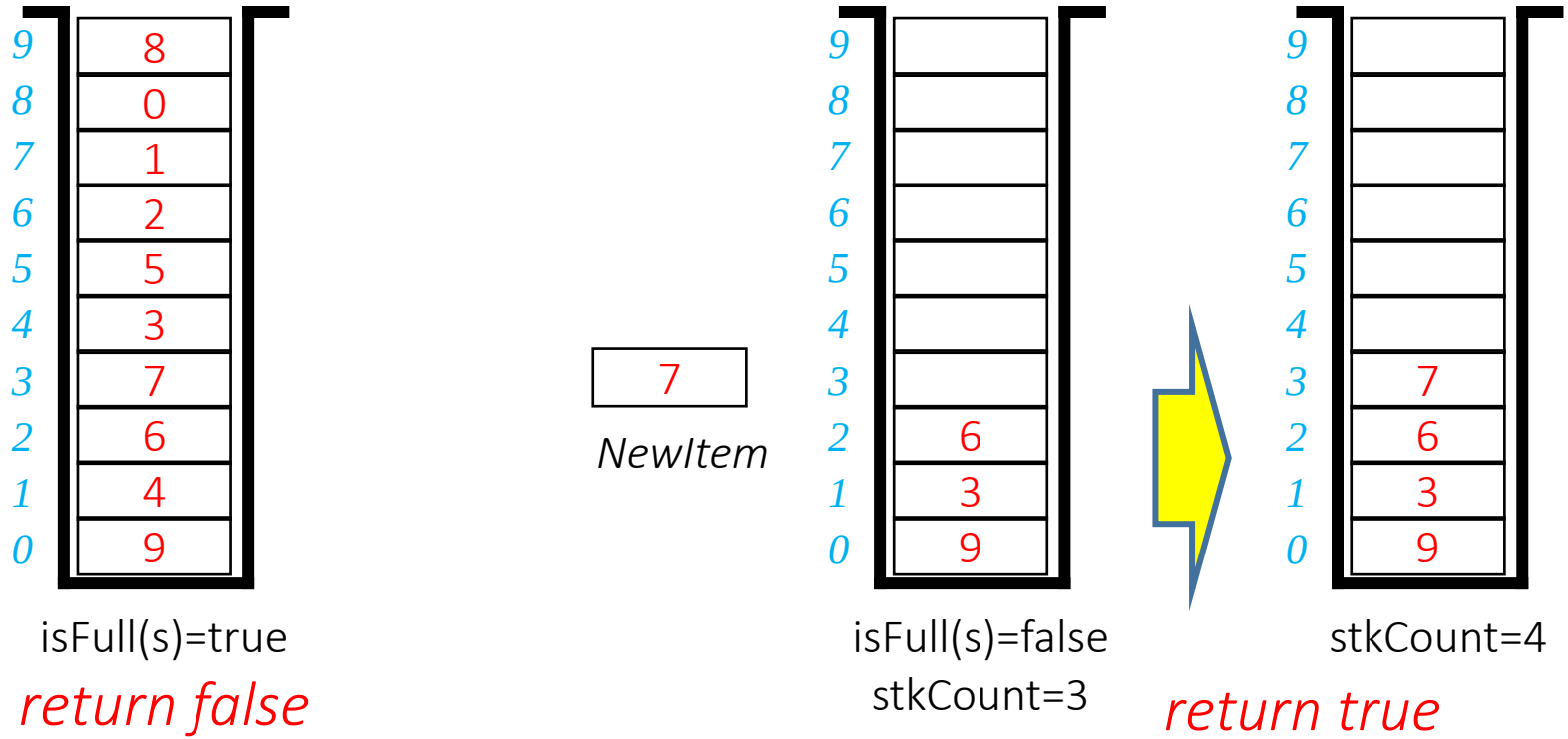
```
bool IsFull(STACK s)
{  /*return true nếu Stack đầy, return false
                                        khi chưa đầy*/
   return (s.StkCount>=s.StkMax);
}
```



5.3. Cài đặt

5.3.5. Thao tác thêm một phần tử vào Stack

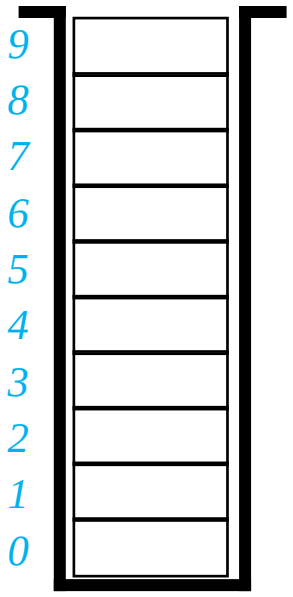
```
bool Push(STACK &s, int NewItem)
{   if (IsFull(s))
        return false;     //stack đầy⇨không thể thêm
    s.StkArray[s.StkCount] = NewItem;
    s.StkCount++;
    return true;          //thêm thành công
}
```



5.3. Cài đặt

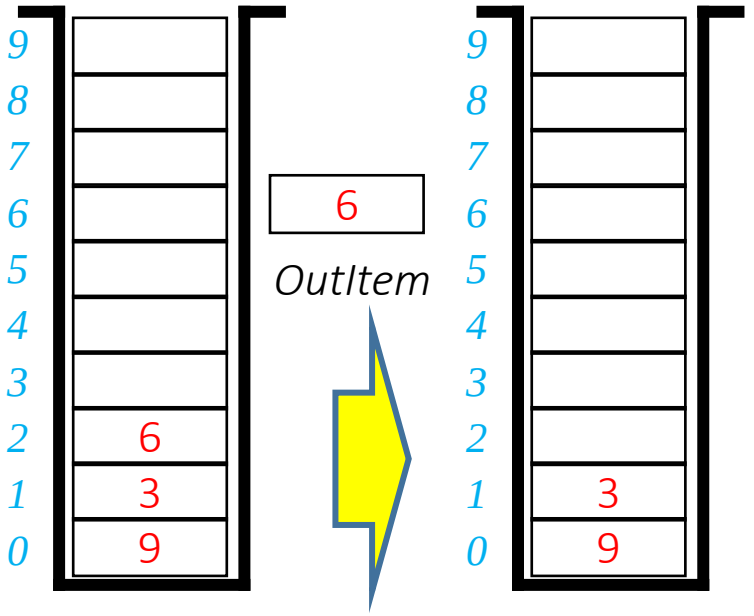
5.3.6. Thao tác lấy ra 1 phần tử từ Stack

```
bool Pop(STACK &s, int &OutItem)
{   if (IsEmpty(s))
        return false;//Stack rỗng ⇨ không lấy được
    s.StkCount--;
    OutItem = s.StkArray[s.StkCount];
    return true;            //lấy ra thành công
}
```



isEmpty(s)=true

return false



isEmpty(s)=false
stkCount=3

stkCount=2

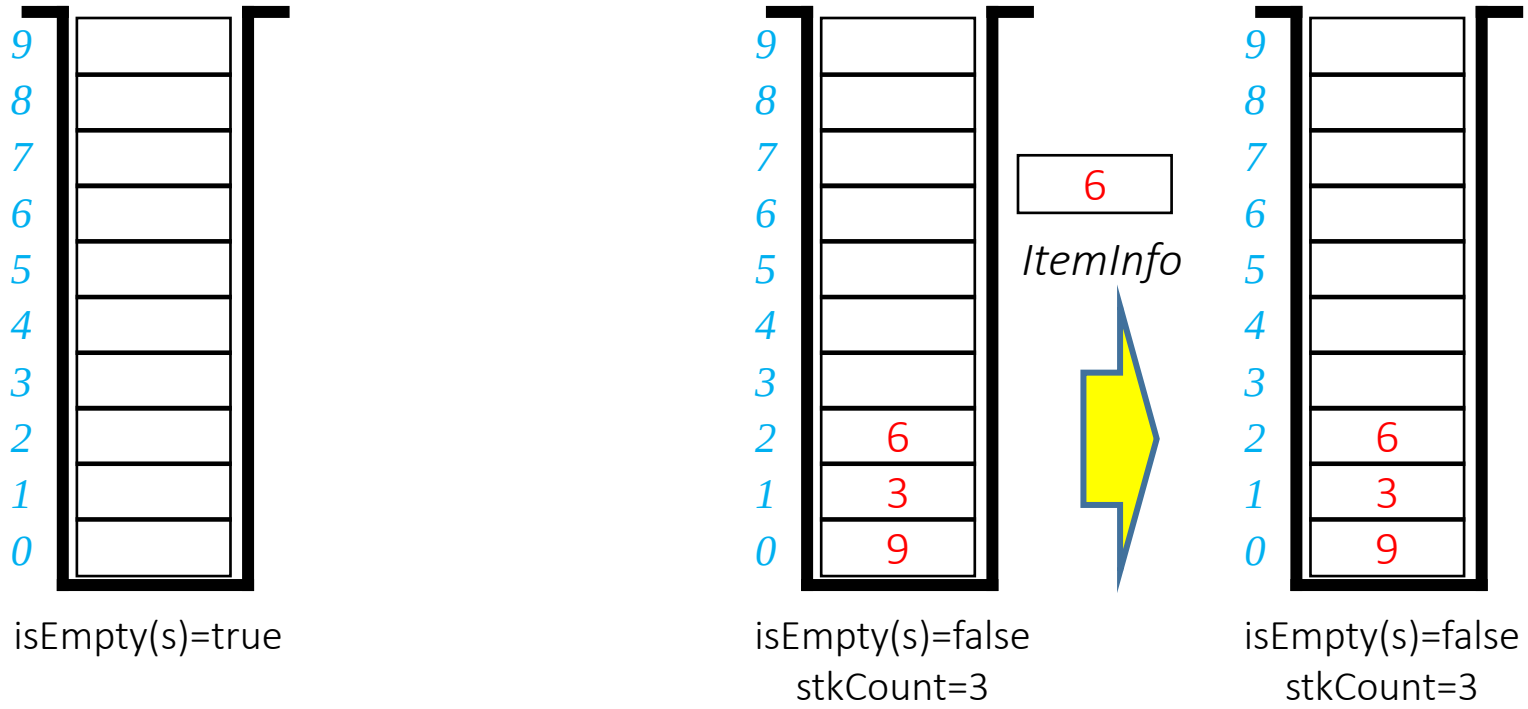
return true

5.3. Cài đặt

5.3.7. Thao tác lấy thông tin của phần tử ở đỉnh Stack

(không làm thay đổi Stack)

```
bool StackTop(STACK s, int &ItemInfo)
{   if (IsEmpty(s))
        return false;//Stack rỗng, không lấy được info
    ItemInfo = s.StkArray[s.StkCount-1];
    return true;     //lấy info thành công
}
```



6. SỬ DỤNG DSLK ĐƠN LÀM STACK

6.1. Khai báo các cấu trúc

- **Khai báo cấu trúc của Node**

```
struct NODE
{    DataType data;
     Node *pNext;
};
```

- **Khai báo cấu trúc của stack**

```
struct NODE* STACK;
```

- **Khai báo biến quản lý stack**

```
STACK s;
```

6.2. Khởi tạo Stack

```
void Init(STACK &s)
{  s.top = NULL;
}
```

6.3. Kiểm tra xem Stack có rỗng không?

```
bool isEmpty (STACK s)
{
   return s.top == NULL;
}
```

6.4. Thêm một phần tử x vào Stack

```
void Push (Stack &s, DataType x)
{    Node *p = new Node;
     if (p==NULL)
     {    printf("Khong du bo nho");
          return;
     }
     p->data = x;
     p->pNext= NULL;
     if (s.top==NULL) // if (Empty(l))
          s.top = p;
     else
     {    p->pNext = s.top;
          s.top = p;
     }
}
```

6.5. Lấy một phần tử ra khỏi Stack

```
DataType Pop (Stack &s)
{
    if (isEmpty(s)
    {  printf("Stack rỗng");
       return NULL;
    }
    DataType x;
    Node *p = s.top;
    p->pNext = NULL;
    s.top = s.top->pNext;
    x = p->data;
    delete p;
    return x;
}
```

7. THỰC HÀNH

Bài tập 7.1: Viết chương trình phân tích 1 số nguyên n thành tích các thừa số nguyên tố theo thứ tự giảm dần

Ví dụ: n=1350 in ra 1350= 5 * 5 * 3 * 3 * 3 * 2

Bài tập 7.2: Viết chương trình cho biết 1 chuỗi có đối xứng hay không?

- ***Gợi ý***: một số chuỗi đối xứng như: "dad", "level", "madam", "radar", "abba"
- Lần lượt đưa từng ký tự của chuỗi str vào stack cho đến khi hết chuỗi str. Hàm return 1 nếu thành công, ngược lại return 0.

```
int PushStringToStack(Stack &stack, char *str)
```

- Hàm trả về chuỗi được lấy ra từ toàn bộ nội dung có trong stack.

```
char* PopFromStack(Stack &stack)
```

7. THỰC HÀNH

Bài tập 7.3:Viết chương trình thực hiện yêu cầu sau, với prototype được đề nghị là:

```
int PushDecToOthers(Stack &stack, int DecValue, int base)
/* + Hàm trả về 1 nếu thành công, ngược lại trả về 0,
   + DecValue là giá trị hệ thập phân, base là cơ số cần đổi.
   + Hàm sẽ gọi hàm Push đã có để bỏ các số từ kết quả chia lấy
dư (%) vào stack*/

void PopDecToOthers(Stack &stack)
 /*hàm sẽ gọi hàm Pop đã có để xuất kết quả ra màn hình*/
```

VIẾT CHƯƠNG TRÌNH SỬ DỤNG STACK

(VỚI CTDL SỬ DỤNG LÀ MẢNG 1 CHIỀU)

Đổi giá trị của 1 số thực không âm từ cơ số **10** sang cơ số **2, 8, 16**

Nhập giá trị hệ thập phân: X

Kết quả xuất ra màn hình:

Giá trị tương ứng của X ở hệ 2 là: Y

Giá trị tương ứng của X ở hệ 8 là: Z

Giá trị tương ứng của X ở hệ 16 là: W

Trong đó

- **X** là giá trị do người dùng nhập vào.
- **Y, Z, W** là kết quả thực hiện của chương trình

Bài tập 7.4: Viết chương trình thực hiện:

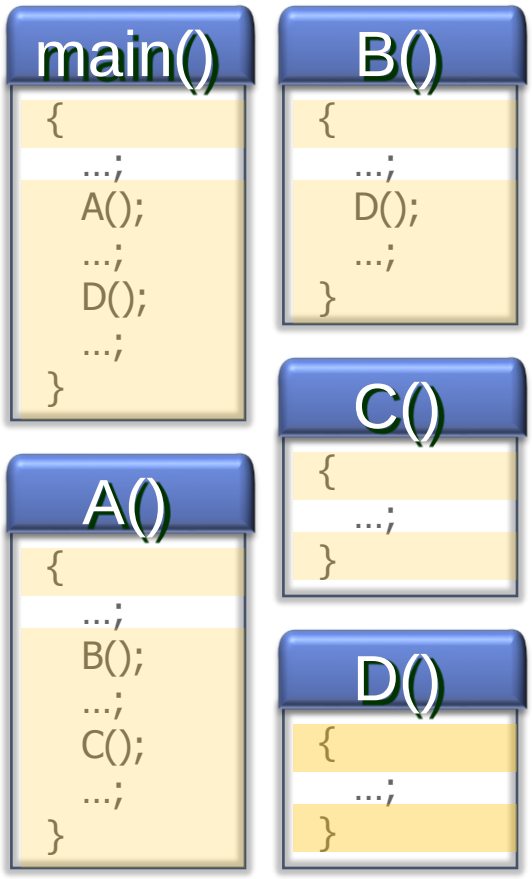
- Cho nhập biểu thức dạng *Infix*.
- Sử dụng stack để chuyển biểu thức vừa nhập sang dạng *Postfix*.
- Thực hiện tính toán giá trị cho biểu thức *Postfix* vừa có

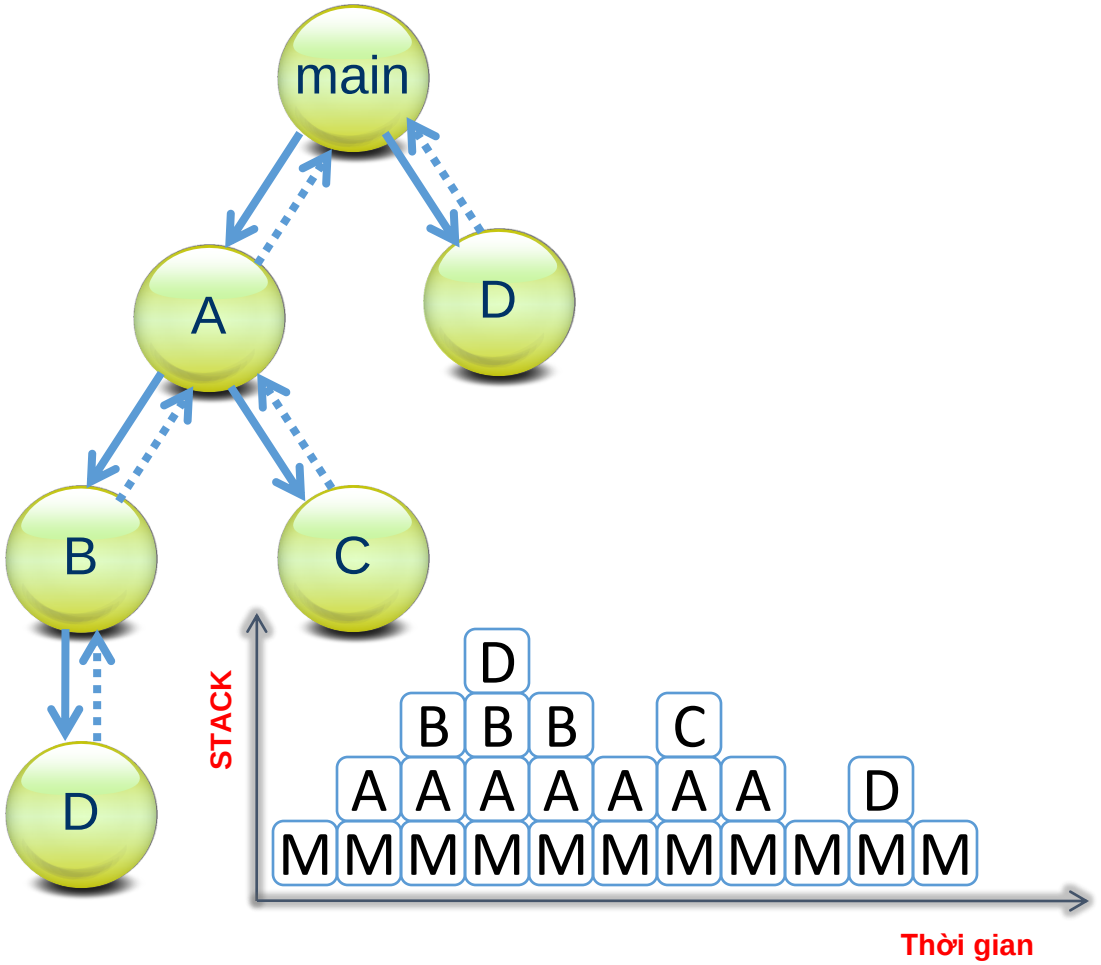
iii. Bài tập 7.5: Cho người dùng nhập các dấu mở và đóng ngoặc, viết hàm kiểm tra xem các dấu ngoặc có được mở và đóng hợp lệ hay không?

- ***Trường hợp 1***: chỉ dùng 1 loại ngoặc đơn **(** và **)**
- ***Trường hợp 2***: dùng cả 3 loại ngoặc: **(,)**; **[,]**; **{, }**

Valid inputs	Invalid Inputs
{}	{(}
({[]})	([(()])
{[]()}	{}[])
[{({}[]({}))}]	[{)}(]}]

8. Cơ chế gọi hàm đệ quy và STACK





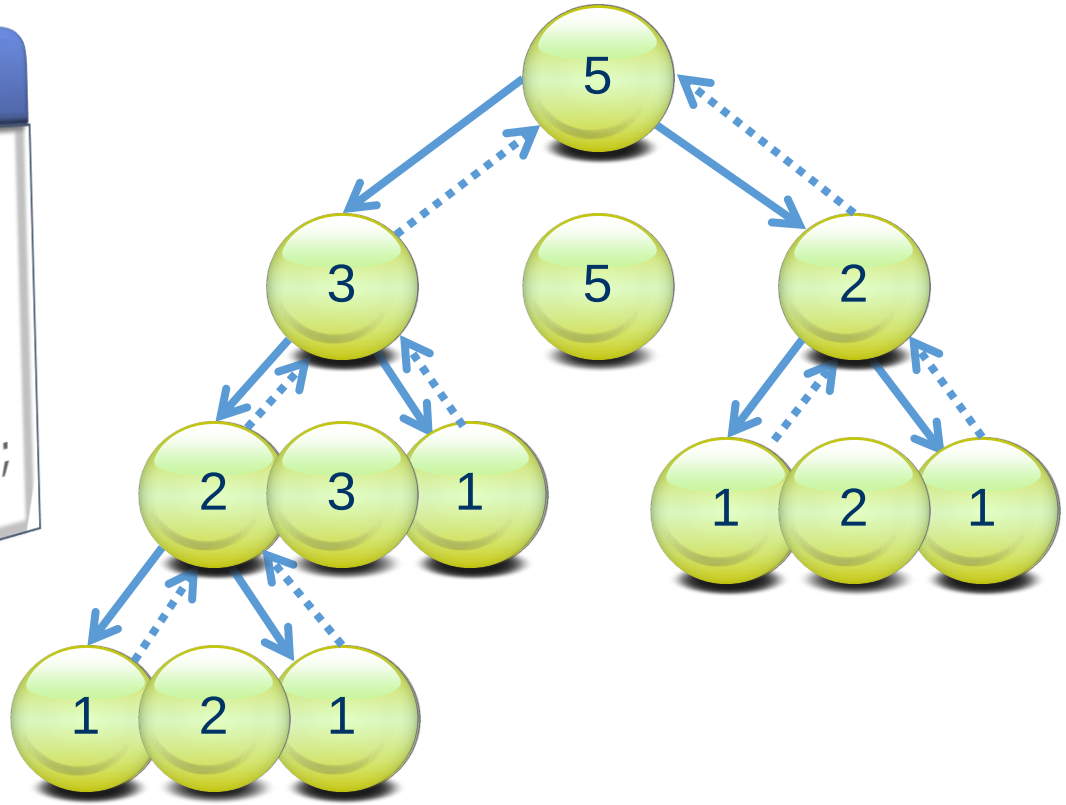
8.1. Nhận xét

- Cơ chế gọi hàm dùng STACK trong C phù hợp cho giải thuật đệ quy vì:
  - Lưu thông tin trạng thái còn dở dang mỗi khi gọi đệ quy.
  - Thực hiện xong một lần gọi cần khôi phục thông tin trạng thái trước khi gọi.
  - Lệnh gọi cuối cùng sẽ hoàn tất đầu tiên.

8.2. Ví dụ gọi hàm đệ quy

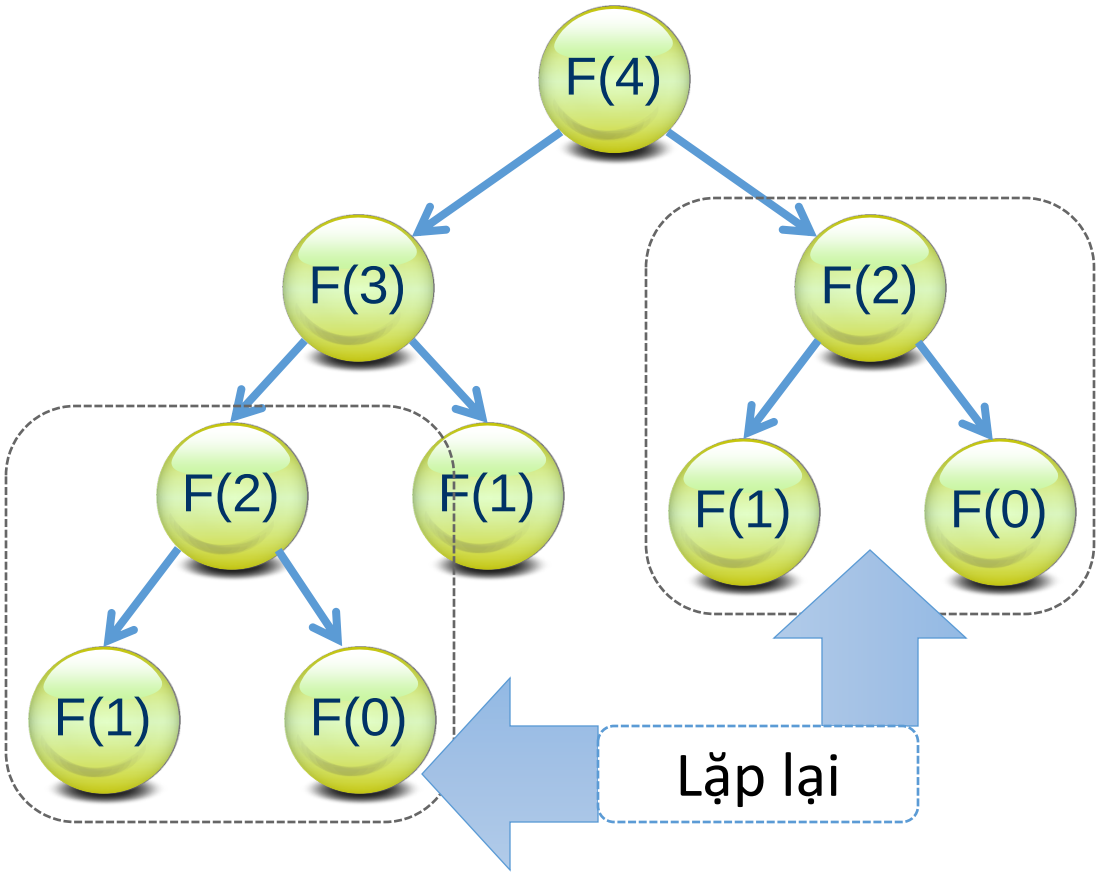
- Tính số hạng thứ 4 của dãy *Fibonacci*

```
long F(int n)
{
    if (n == 0)
        return 1;
    if (n == 1)
        return 1;
    return F(n-1) + F(n-2);
}
```



8. Cơ chế gọi hàm đệ quy và STACK

Ví dụ cây đệ quy *Fibonacci*



8.3. Phân tích giải thuật đệ quy

- Sử dụng cây đệ quy (*recursive tree*)
  - Giúp hình dung bước phân tích và thế ngược.
  - Bước phân tích: đi từ trên xuống dưới.
  - Bước thế ngược đi từ trái sang phải, từ dưới lên trên.
  - Ý nghĩa
    - Chiều cao của cây ⇔ Độ lớn trong STACK.
    - Số nút ⇔ Số lời gọi hàm.

8.4. Khử đệ quy

- Khái niệm
  - Đưa các bài toán đệ quy về các bài toán không sử dụng đệ quy.
  - Thường sử dụng vòng lặp hoặc STACK tự tạo.

Ví dụ: Tính tổng S = 1 + 2 + … + n

Đệ quy	*Hàm không đệ quy*

`int Tong(int n)`
`{`
`  if (n==1)`
`  return 1;`
`  return Tong(n-1)+n;`
`}`
	`int Tong(int n)`
	`{  int S=0;`
	`  for (int i=1;i<=n;i++)`
	`  S+=1;;`
	`  return S;`
	`}`


Q&A
How
Why
When
Where
What
Who