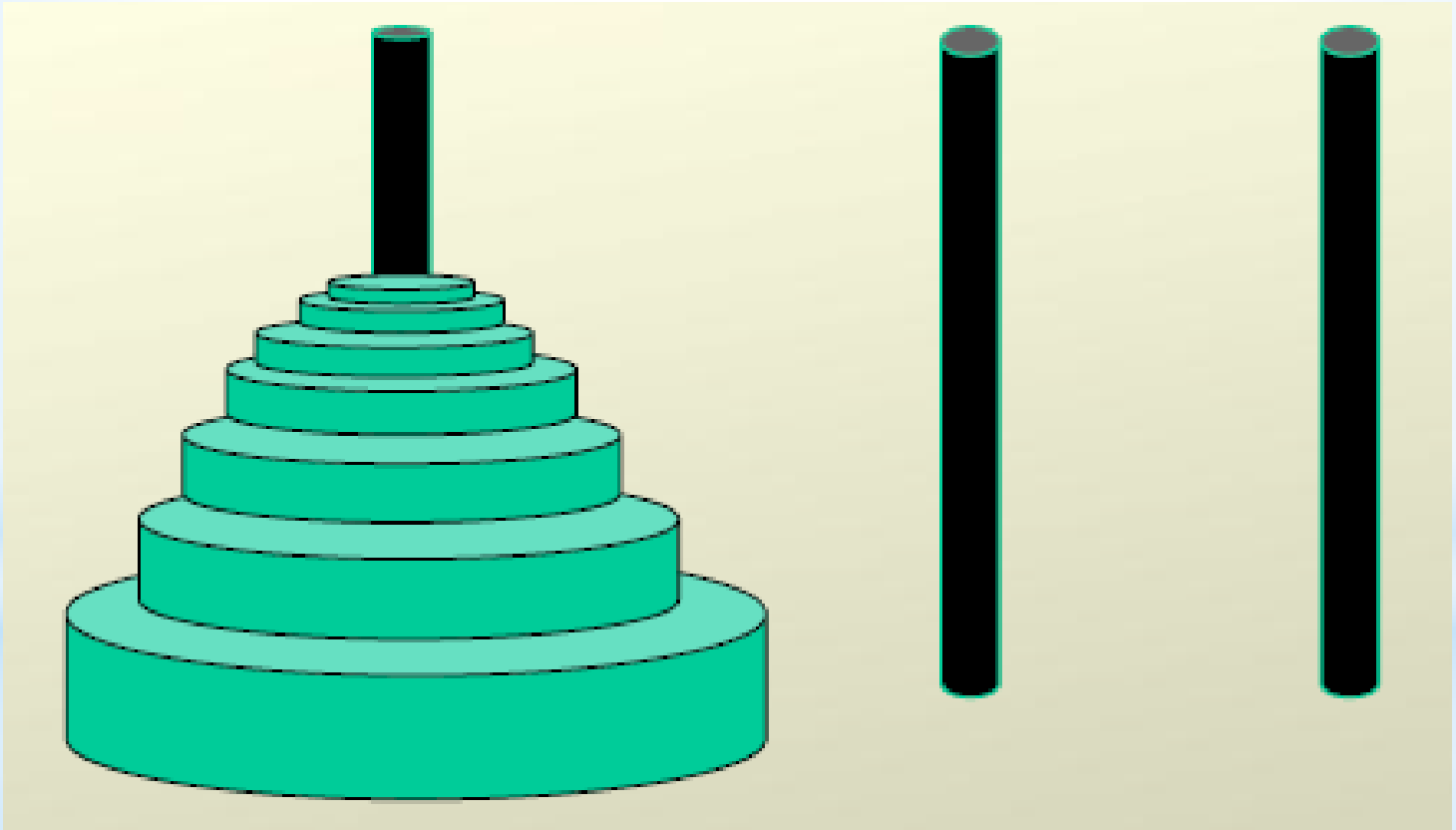
3.5. Ngăn xếp (Stack)

3.5.1. Định nghĩa. Tập hợp các node thông tin được tổ chức liên tục hoặc rời rạc nhau trong bộ nhớ và thực hiện theo cơ chế FILO (First – In – Last – Out).

Ví dụ. Bài toán tháp Hà Nội.



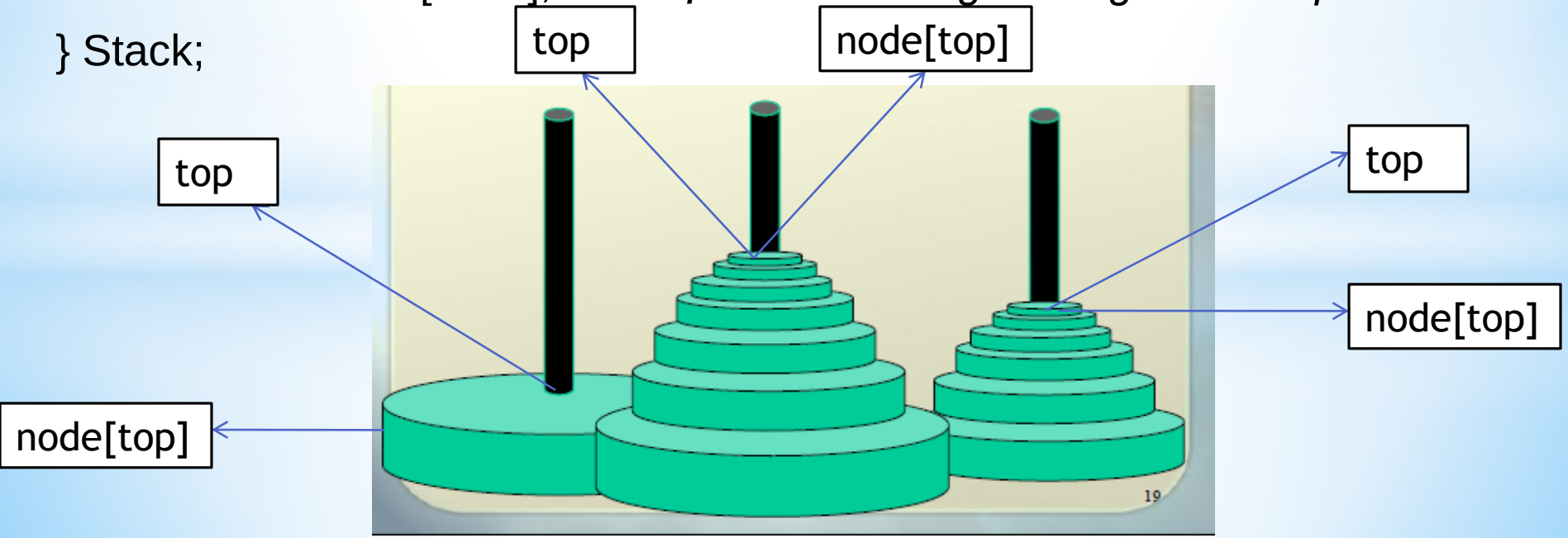
3.5.2. Biểu diễn dựa vào mảng

Có hai phương pháp biểu diễn ngăn xếp:

- **Biểu diễn liên tục**: các phần tử dữ liệu của ngăn xếp được lưu trữ liên tục nhau trong bộ nhớ (Mảng).
- **Biểu diễn rời rạc**: các phần tử dữ liệu của ngăn xếp được lưu trữ rời rạc nhau trong bộ nhớ (Danh sách liên kết).

Ví dụ. Biểu diễn ngăn xếp dựa vào mảng.

```
typedef struct {
        int     top; //Đỉnh đầu của stack nơi diễn ra mọi thao tác
        int     node[MAX]; //Dữ liệu lưu trữ trong stack gồm MAX phần tử
} Stack;
```

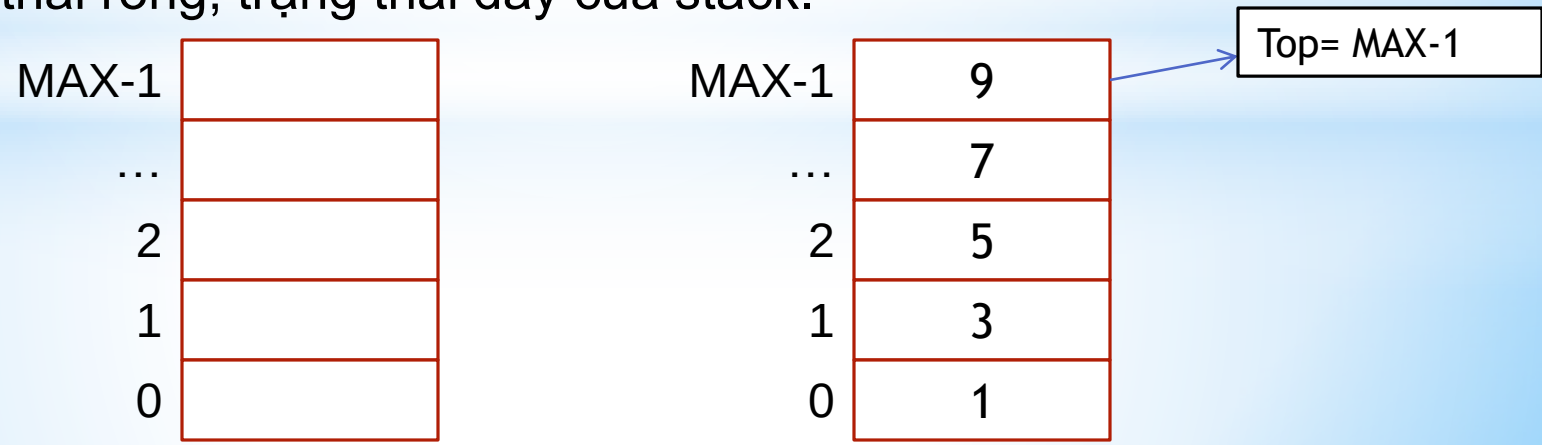


3.5.3. Các thao tác trên stack dựa vào mảng

Các thao tác xây dựng trên stack theo cơ chế Last-In-First-Out bao gồm:

- **Kiểm tra tích rỗng của stack** (*Empty*(*stack s*))**.** Khi ta muốn lấy dữ liệu ra khỏi ngăn xếp thì nó chỉ được thực hiện khi và chỉ khi ngăn xếp không rỗng.
- **Kiểm tra tính đầy của ngăn xếp** (*Full*(*stack s*))**.** Khi ta muốn đưa dữ liệu vào ngăn xếp thì nó chỉ được thực hiện khi và chỉ khi ngăn xếp chưa tràn.
- **Đưa dữ liệu vào ngăn xếp** (*Push*(*stack s, item x*))**.** Thao tác này chỉ được thực hiện khi và chỉ khi ngăn xếp chưa tràn.
- **Đưa dữ liệu vào ngăn xếp** (*Pop*(*stack s*)**.** Thao tác này chỉ được thực hiện khi và chỉ khi ngăn xếp không rỗng.

Ví dụ. Trạng thái rỗng, trạng thái đầy của stack.



Kiểm tra tính rỗng của stack:

Tất cả các thao tác trên stack chỉ được thực hiện tại vị trí con trỏ top. Vì vậy ta qui ước tại vị trí top = -1 stack ở trạng thái rỗng. Thao tác được thực hiện như sau:

```
typedef struct {
        int    top; //con trỏ top
        int    node[MAX]; //Các node của stack
} stack;
int     Empty( stack *s ) { //s là con trỏ đến stack
        if ( stack ->top == -1 )  // Nếu top =-1
            return (1); //Hàm trả lại giá trị đúng
        return(0); //Hàm trả lại giá trị sai
}
```

Kiểm tra tính đầy của stack:

Khi ta muốn lấy dữ liệu ra khỏi ngăn xếp thì ngăn xếp phải chưa tràn. Vì biểu diễn dựa vào mảng, do đó tại vị trí top = MAX -1 thì stack ở trạng thái đầy. Thao tác được thực hiện như sau:

```
typedef struct {
        int    top; //con trỏ top
        int    node[MAX]; //Các node của stack
} stack;
int     Full( stack *s ) { //s là con trỏ đến stack
        if ( stack ->top == MAX-1 )  // Nếu top = MAX -1
            return (1); //Hàm trả lại giá trị đúng
        return(0); //Hàm trả lại giá trị sai
}
```

Đưa dữ liệu vào stack:

Khi muốn đưa dữ liệu vào ngăn xếp thì ta phải kiểm tra ngăn xếp có đầy (tràn) hay không? Nếu ngăn xếp chưa đầy, thao tác sẽ được thực hiện. Nếu ngăn xếp đã đầy, thao tác không được thực hiện. Thao tác được thực hiện như sau:

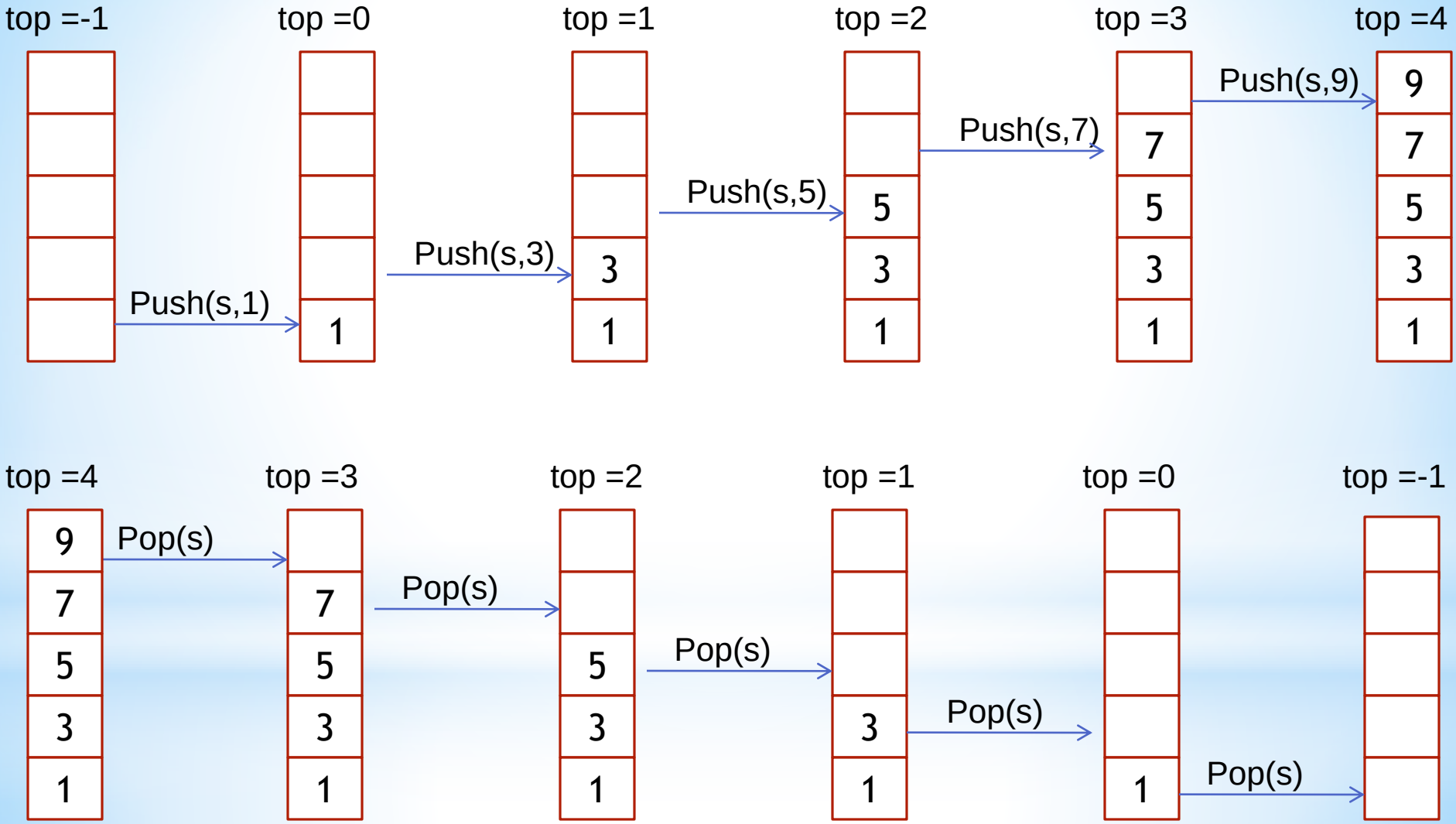
```
typedef struct {
        int    top; //con trỏ top
        int    node[MAX]; //Các node của stack
} stack;
void     Push( stack *s, int   x ) { //x là node cần thêm vào stack
        if ( !Full (s))  // Nếu stack chưa tràn
            s ->top = (s ->top) +1; //Tăng con trỏ top lên 1 đơn vị
            tode[s ->top] = x; //Lưu trữ x tại vị trí top
        }
        else <thông báo tràn stack>;
}
```

Lấy dữ liệu ra khỏi stack:

Khi muốn lấy dữ liệu ra khỏi ngăn xếp thì ta phải kiểm tra ngăn xếp có rỗng hay không? Nếu ngăn xếp không rỗng, thao tác sẽ được thực hiện. Nếu ngăn xếp rỗng, thao tác không được thực hiện. Thao tác được thực hiện như sau:

```
typedef struct {
        int    top; //con trỏ top
        int    node[MAX]; //Các node của stack
} stack;
int     Pop( stack *s ) { //s là con trỏ đến stack
        if ( !Empty (s)) { // Nếu stack không rỗng
            x = Node[s ->top]; //x là nội dung node bị lấy ra
            s ->top = (s ->top) -1; //giảm con trỏ top 1 đơn vị
            return (x); //Trả lại x là node bị loại bỏ
        }
        else { <thông báo stack rỗng>; return (∞); }
}
```

Ví dụ. Cho stack lưu trữ tối đa 5 phần tử. Bắt đầu thực hiện tại trạng thái rỗng.



3.5.4. Các thao tác trên stack dựa danh sách liên kết đơn

Xây dựng Stack bằng danh sách liên kết đơn được thực hiện như sau:

Stack = { DSLK đơn + [<Add-Top:(Push)>; <Del-Top: (Push)>] }

Stack = { DSLK đơn + [<Add-Bottom:(Push)>; <Del-Bottom: (Push)>] }

Lớp các thao tác trên stack được xây dựng như sau:

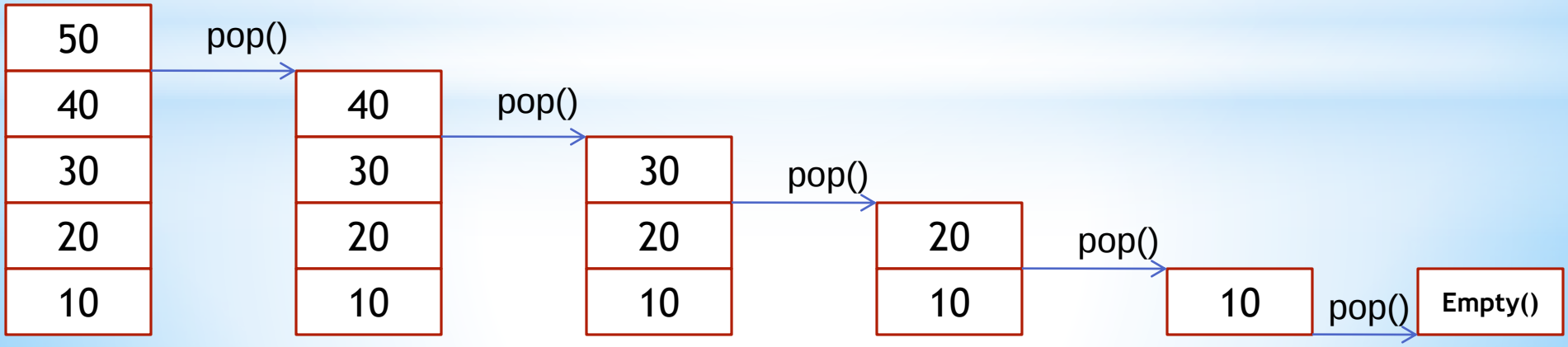
```
struct node{//Biểu diễn stack
        int info; //Thành phần thông tin của node
        struct node *link; //Thành phần con trỏ của node
}*Stack;
class stack_list{ //Mô tả lớp stack_list
        public:
                node *push(node *, int); //Thêm node
                node *pop(node *); //Loại bỏ node
                void traverse(node *);//Duyệt stack
                stack_list(){   Stack = NULL; }  //Constructor của lớp
};
```

```
node *stack_list::push(node *Stack, int item){ //Thêm node vào stack
        node *tmp; tmp = new (struct node);
        tmp->info = item; tmp->link = Stack; Stack = tmp;
        return Stack;
}
node *stack_list::pop(node *Stack){  node *tmp;
        if (Stack == NULL)  cout<<"Stack rỗng"<<endl;
        else  {  tmp = Stack;
                cout<<“Node đã bị loại bỏ: "<<tmp->info<<endl;
                Stack = Stack->link; free(tmp);
        }
        return Stack;
}
void stack_list::traverse(node *Stack){
        node *ptr; ptr = Stack;
        if (Stack == NULL)  cout<<"Stack rỗng"<<endl;
        else  {  cout<<“Các phần tử của stack :"<<endl;
                while (ptr != NULL)   {
                        cout<<ptr->info<<endl; ptr = ptr->link;
                }
        }
}
```

CÁC THAO TÁC TRÊN STACK SỬ DỤNG STL		
STT	Thao tác	Ý nghĩa
1	S.push(item)	Đưa item vào ngăn xếp S
2	S.pop()	Loại node trong ngăn xếp S
3	S.top()	Truy cập (lấy) phần tử ở đầu ngăn xếp S
4	S.size()	Kích cỡ ngăn xếp được tính bằng số lượng phần tử
5	S.empty()	Kiểm tra tính rỗng của ngăn xếp.
6	S.Swap(S1)	Tráo đổi các phần tử của S1 thành S và S thành S1.

```
//Cài đặt stack bằng STL:
#include <iostream>
#include <stack>
using namespace std;
int main(void){
        stack <int> s; //khai báo stack thích nghi với tùy biến int
        for(int i=1; i<=5; i++)  s.push(i*10); //đưa i*10 vào stack
        cout<<"Kích cỡ stack:"<<s.size()<<endl;// s.size() = 10
        while(!s.empty()){ //lặp đến khi stack rỗng
                int t = s.top();cout<<t<<" "; //lấy phần tử đầu stack
                s.pop(); //đưa phần tử đầu tiên ra khỏi stack
        }
}
```

Hình 1: push(1*10), push(2*10), push(3*10), push(4*10), push(5*10)



3.5.5. Ứng dụng của ngăn xếp

- Xây dựng các giải thuật đệ qui.
- Khủ bỏ các giải thuật đệ qui.
- Biểu diễn tính toán.
- Duyệt cây, duyệt đồ thị…

Ví dụ 1. Biểu diễn các biểu thức thức số học dạng hậu tố

- a + b $\Leftrightarrow$ a b +
- a - b $\Leftrightarrow$ a b -
- a * b $\Leftrightarrow$ a b *
- a / b $\Leftrightarrow$ a b /
- (P) $\Leftrightarrow$ P

Ví dụ.

$$(a+b*c)-(a/b+c)=$$
$$=(a+bc*)-(ab/+c)$$
$$=(abc*+)-(ab/c+)$$
$$=abc*+-ab/c+$$
$$=abc*+ab/c+-$$

Ví dụ 1. Tính toán biểu thức số học hậu tố

- a + b $\Leftrightarrow$ a b +
- a - b $\Leftrightarrow$ a b -
- a * b $\Leftrightarrow$ a b *
- a / b $\Leftrightarrow$ a b /
- (P) $\Leftrightarrow$ P

Ví dụ.

$$(a+b*c)-(a/b+c)=$$

$$=(a+bc*)-(ab/+c)$$

$$=(abc*+)-(ab/c+)$$

$$=abc*+-ab/c+$$

$$=abc*+ab/c+-$$

Thuật toán chuyển đổi biểu thức trung tố P thành biểu thức hậu tố?

Bước 1 (Khởi tạo)**:** *stack = ∅; Out = ∅;*

Bước 2 (Lặp) :

For each x∈P do

2.1. Nếu x =‘ (‘ : Push(stack, x);

2.2. Nếu x là toán hạng: x⇒Out;

*2.3. Nếu x ∈ { +, -, *, / }*

y = get(stack);

a) Nếu priority(x)≥priority(y): Push(stack, x);

b) Nếu priority(x)<priority(y):

y = Po(stack); y⇒Out; Push(stack, x);

c) Nếu stack = ∅: Push(stack, x);

2.4. Nếu x =’):

y = Pop(stack);

While (y!=‘(’) do

y⇒Out; y = Pop(stack);

EndWhile;

EndFor;

Bước 3(Hoàn chỉnh biểu thức hậu tố)*:*

While (*stack ≠ ∅*) *do*

y = Pop(stack); y⇒Out;

EndWhile;

Bước 4(Trả lại kết quả)*:*

Return(Out).

Kiểm nghiệm thuật toán: P = $(a+b*c)-(a/b+c)$

x∈P	Bước	Stack	Out
x =‘(‘	2.1	(	∅
x =a	2.2	(	a
x =+	2.3.a	(+	a
x =b	2.2	(+	a b
x =*	2.3.a	(+ *	a b
x = c	2.2	(+ *	a b c
x =‘)‘	2.3	∅	a b c * +
x =-	2.2.c	-	a b c * +
x =‘(‘	2.1	- (	a b c * +
x =a	2.2	- (	a b c * + a
x =/	2.2.a	- (/	a b c * + a
x =b	2.2	- (/	a b c * + a b
x =+	2.3.b	- (+	a b c * + a b/
x =c	2.2	- (+	a b c * + a b/c
x =‘)‘	2.4	∅	a b c * + a b/c + -
P = a b c * + a b/c + -			

Thuật toán tính toán giá trị biểu thức hậu tố?

Bước 1 (Khởi tạo)**:**

stack = ∅;

Bước 2 (Lặp) :

For each x∈P do

2.1. Nếu x là toán hạng:

Push(stack, x);

*2.2. Nếu x ∈ { +, -, *, / }*

a) TH2 = Pop(stack, x);

b) TH1 = Pop(stack, x);

c) KQ = TH1 ⊗ TH2;

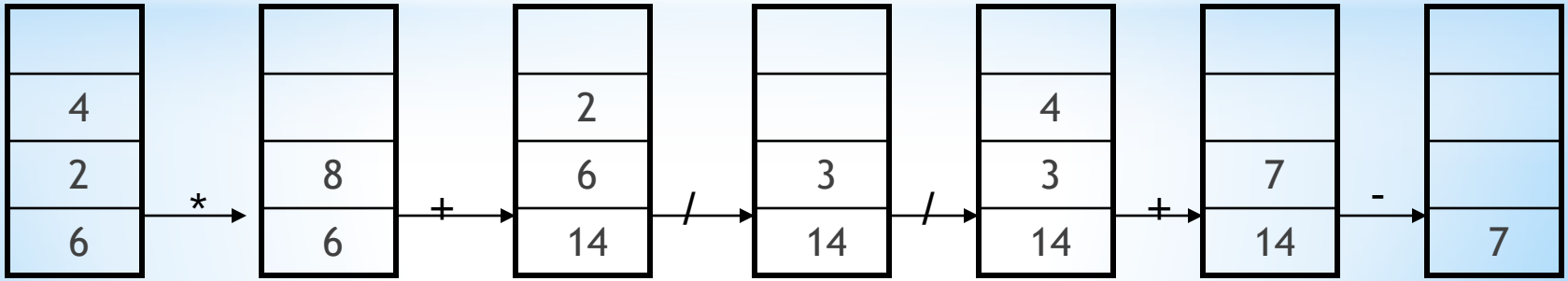
d) Push (stack, KQ);

EndFor;

Bước 4(Trả lại kết quả)*:*

Return(Pop(stack)).

Ví dụ: P = 6 2 4 * + 6 2 / 4 + -



BÀI TẬP

Bracket Numbers (Flipkart). Cho biểu thức exp độ dài n chứa đựng một số ký tự '(', ')'. Hãy in ra số thứ tự của các cặp '(', ')' khi phân tích biểu thức.

Input:	**Output:**
(a + (b *c)) + (d/e)	1 2 2 1 3 3
((()) (()))	1 2 3 3 2 4 5 5 4 1

Thuật toán Index-bracket(exp, n) {*//n là độ dài biểu thức exp*

Bước 1(khởi tạo):

Left = 1; stack <int> right; *//chỉ số đầu tiên bên trái là 1*

Bước 2 (lặp):

```
For(i=0; i<n; i++) { //duyệt từ trái qua phải exp
    If ( exp[i] ==‘(‘) {//nếu exp[i] là ‘(‘
        <đưa ra chỉ số left>; right.push(left); //đưa chỉ số left vào stack
        Left++; //chỉ số tiếp theo được tăng lên 1
    }
    Else if (exp[i] =‘)’ ) { //nếu exp[i] là ‘)’
        <đưa ra chỉ số tương ứng là right.top()>;
        right.pop();// đưa chỉ số ra khỏi ngăn xếp
    }
}//endfor
}
```

BÀI TẬP

Prefix to Infix Conversion. Có ba dạng biểu diễn cho các biểu thức số học và logic:

Infix (trung tố): Biểu diễn biểu thức dưới dạng trung tố là phép biểu diễn biểu thức trong đó phép toán được đặt giữa hai toán hạng. Ví dụ (A+B) * (C-D).

Prefix (tiền tố): Biểu diễn biểu thức dưới dạng tiền tố là phép biểu diễn biểu thức trong đó phép toán được đặt trước hai toán hạng. Ví dụ *+AB-CD (tương ứng với biểu thức trung tố (A+B)*(C-D).

Postfix (hậu tố): Biểu diễn biểu thức dưới dạng hậu tố là phép biểu diễn biểu thức trong đó phép toán được đặt sau hai toán hạng. Ví dụ AB+CD-* (tương ứng với biểu thức trung tố (A+B)*(C-D).

Hãy viết chương trình chuyển đổi biểu thức biểu diễn dưới dạng tiền tố về dạng trung tố.

Input:	**Output:**
*+AB-CD	((A+B)*(C-D))
*-A/BC-/AKL	((A-(B/C))*((A/K)-L)

Thuat toan PreToInfix (Pre_exp) {// *Pre-exp là một string tiền tố*

Bước 1(khởi tạo):

Stack <string> s; *//tạo stack s thích nghi với tùy biến string*

n = Pre_exp.zise(); *//lấy n là độ dài Pre_exp*

Bước 2 (lặp):

For(i=n-1; i>=0; i--) { *//duyệt từ phải qua trái Pre_exp*

X = Pre_exp[i]; // l*ấy X là Pre_exp[i]*

If (isOperator(X)) { *//nếu X là phép toán*

String Op1 = s.top(); s.pop(); *//đưa toán hạng 1 ra khỏi stack*

String Op2 = s.top(); s.pop(); *//đưa toán hạng 2 ra khỏi stack*

String temp =“(“ +Op1 + X + OP2 + “)”; *//thành lập string temp*

s.push(temp); *//đưa temp trở lại stack*

}

Else { *//nếu X là toán hạng*

s.push(string(1), X); *//đưa X vào stack với kiểu string độ dài 1*

}

}

Return (s.top()); *//phần tử cuối cùng chính là biểu thức trung tố*

}

//Chú ý:

// isOperator(X) = true nếu x là phép toán ngược lại isOperator(X) = false

BÀI TẬP

Prefix to Infix Conversion. Có ba dạng biểu diễn cho các biểu thức số học và logic:

Infix (trung tố): Biểu diễn biểu thức dưới dạng trung tố là phép biểu diễn biểu thức trong đó phép toán được đặt giữa hai toán hạng. Ví dụ (A+B) * (C-D).

Prefix (tiền tố): Biểu diễn biểu thức dưới dạng tiền tố là phép biểu diễn biểu thức trong đó phép toán được đặt trước hai toán hạng. Ví dụ *+AB-CD (tương ứng với biểu thức trung tố (A+B)*(C-D).

Postfix (hậu tố): Biểu diễn biểu thức dưới dạng hậu tố là phép biểu diễn biểu thức trong đó phép toán được đặt sau hai toán hạng. Ví dụ AB+CD-* (tương ứng với biểu thức trung tố (A+B)*(C-D).

Hãy viết chương trình chuyển đổi biểu thức biểu diễn dưới dạng tiền tố về dạng trung tố.

Input:	**Output:**
*+AB-CD	((A+B)*(C-D))
*-A/BC-/AKL	((A-(B/C))*((A/K)-L)

BÀI TẬP

1. Ta gọi NGE(i) của một mảng A[] là phần tử lớn hơn A[i] đầu tiên bên phải A[i]; NGE(i) = -1 nếu i là phần tử cuối cùng của mảng hoặc bên phải A[i] không có phần tử nào lớn hơn A[i]. Cho mảng A[] gồm n phần tử, hãy in ra NGE(i) của mỗi phần tử với độ phức tạp thời gian O(n).

Ví dụ:

Input:

```
2
4
13  7  6  12
5
8  12  9  7  5
```

Output:

```
-1  12  12  -1
12  -1  -1  -1  -1
```

Thuật toán:

```
void NGE(ll A[], int n) { //đưa ra phần tử lớn hơn tiếp theo
    Bước 1: khởi tạo
        stack<ll> s; //khởi tạo stack s
        ll A1[n]; //tạo mảng lưu các phần tử lớn hơn
    Bước 2: bước lặp
        for (ll i = n - 1; i >= 0; i--) {//duyệt từ phải qua trái mảng A[]
                //lặp nếu stack khác rỗng và s.top<A[i]
                while (!s.empty() && s.top() < A[i])
                        s.pop();//loại phần tử ra khỏi stack
                if (s.empty())  //nếu stack rỗng
                        A1[i] = -1;//không có phần tử lớn hơn
                else //phần tử NGE của A[i] là s.top()
                        A1[i] = s.top(); //thiết lập phần tử lớn hơn cho A[i]
                s.push(A[i]); //đưa A[i] vào ngăn xếp
        }
    Bước 3: đưa ra kết quả
        for (ll i = 0; i < n; i++)
                cout << arr1[i] <<" ";
        cout<<endl;
}
// A[] = { 13, 7, 6, 12 }
```