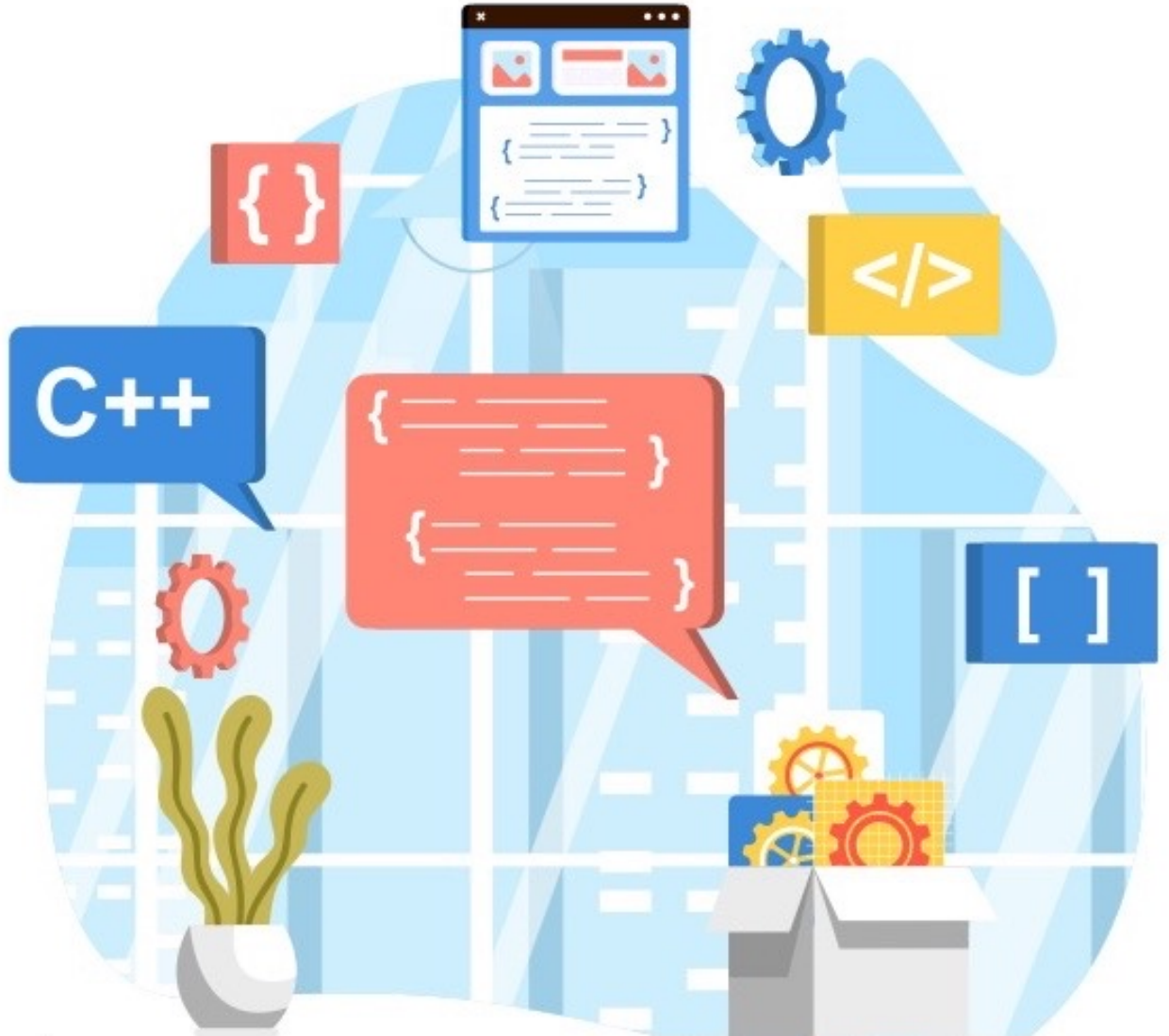
NGĂN XẾP VÀ HÀNG ĐỢI



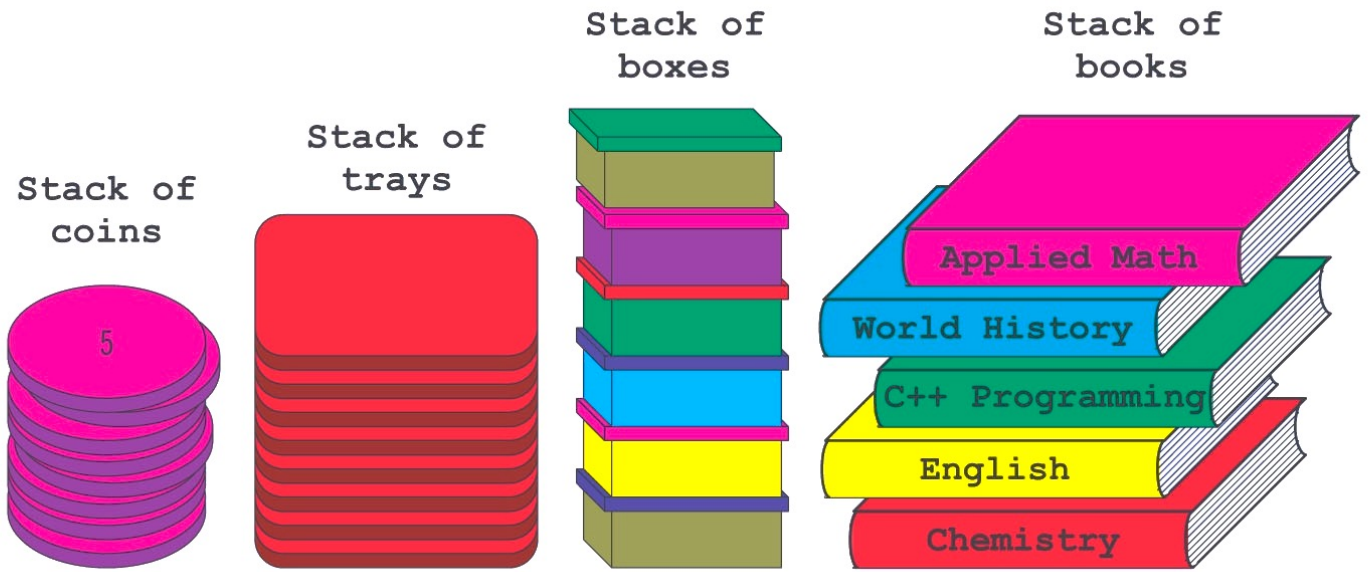
VŨ NGỌC THANH SANG

KHOA CÔNG NGHỆ THÔNG TIN

ĐẠI HỌC SÀI GÒN

I - Stacks

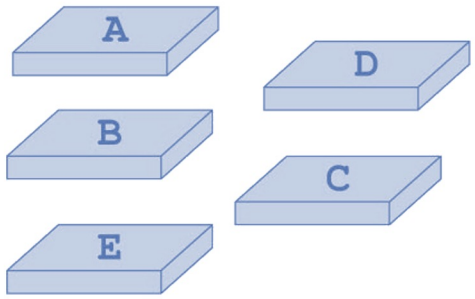
- Ngăn xếp (stacks): là một cấu trúc dữ liệu mà trong đó các phần tử chỉ được thêm/xóa từ đỉnh (top) của ngăn xếp.
- Cấu trúc dữ hoạt động theo nguyên lý vào sau ra trước (Last-In-First-Out, LIFO)



I - Stacks

- Các thao tác trên ngăn xếp
  - initializeStack: khởi tạo ngăn xếp
  - isEmpty: kiểm tra xem ngăn xếp có rỗng hay không
  - isFull: kiểm tra xem ngăn xếp đã đầy hay chưa
  - push: thêm một phần tử vào đỉnh của ngăn xếp
  - top: trả về giá trị của phần tử ở đỉnh của ngăn xếp.
  - pop: xóa phần tử ở đỉnh của ngăn xếp

I - Stacks



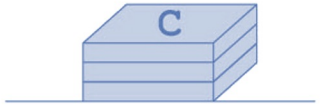
Empty stack

Push Box A

(a)

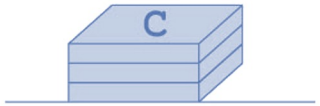
Push Box B

(b)



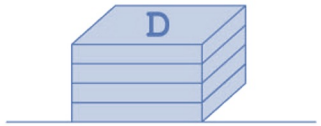
Push Box C

(c)



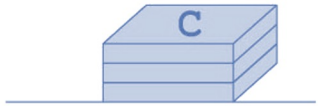
Peek at the top element

(d)



Push Box D

(e)



Pop stack

(f)

I – Stacks

- Ngăn xếp có thể được tạo và triển khai dựa trên một trong 3 cách sau

1. Sử dụng mảng 1 chiều (class)
2. Sử dụng DSLK (struct)
3. Sử dụng thư viện mẫu chuẩn (standard template library) C++

I – Stacks – 1D Array Implementation

- Khởi tạo ngăn xếp

```
#include<iostream>
#define SIZE 100
using namespace std;

class STACK{
    private:
        int num[SIZE];
        int top;
    public:
        STACK();
        int push(int);
        int pop();
        bool isEmpty();
        bool isFull();
        void displayItems();
};
STACK::STACK(){
    top=-1;
}
```

I – Stacks – 1D Array Implementation

- Kiểm tra xem ngăn xếp có rỗng hay không

```
bool STACK::isEmpty()
{
    if(top==-1)
        return true;
    else
        return false;
}
```

I – Stacks – 1D Array Implementation

- Kiểm tra xem ngăn xếp đã đầy chưa

```
bool STACK::isFull()
{
    if(top==(SIZE-1))
        return true;
    else
        return false;
}
```

I – Stacks – 1D Array Implementation

- Thêm phần tử vào đỉnh ngăn xếp

```
int STACK::push(int n)
{
    if(isFull())
    {
        return -9999;
    }
    ++top;
    num[top]=n;
    return n;
}
```

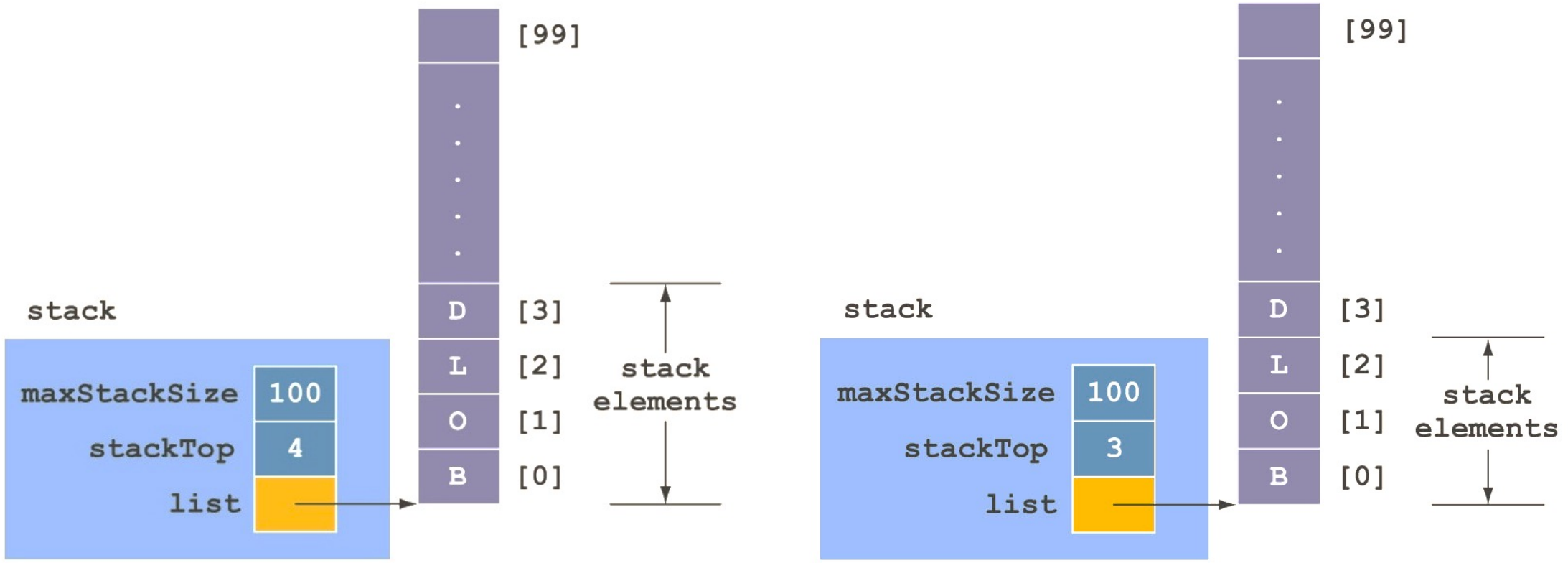
I – Stacks – 1D Array Implementation

- Xoá phần tử ở đỉnh ngăn xếp

```
int STACK::pop()
{
    int temp;
    if(isEmpty())
        return -9999;
    temp=num[top];
    --top;
    return temp;
}
```

I – Stacks – 1D Array Implementation

- Xoá phần tử ở đỉnh ngăn xếp



I – Stacks – 1D Array Implementation

- In các phần tử ở đỉnh ngăn xếp (từ trên xuống dưới)

```
void STACK::displayItems()
{
    int i;
    cout<<"STACK is: ";
    for(i=(top); i>=0; i--)
        cout<<num[i]<<" ";
    cout<<endl;
}
```

I – Stacks – Linked List Implementation

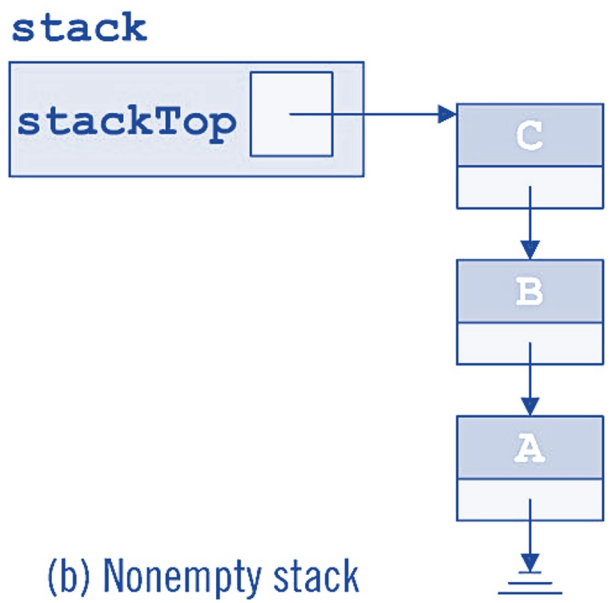
- Mảng chỉ cho phép thao tác với số lượng phần tử cố định
- Nếu số phần tử được thêm vào ngăn chứa dạng mảng vượt quá kích thước của mảng sẽ xảy ra hiện tượng tràn ngăn chứa.
- Sử dụng DSLK có thể giải quyết được vấn đề này.

I – Stacks – Linked List Implementation

- Khởi tạo ngăn chứa dạng DSLK

```
struct Node
{
    int data;
    struct Node *next;
};
struct Node* top = nullptr;
```

(a) Empty stack



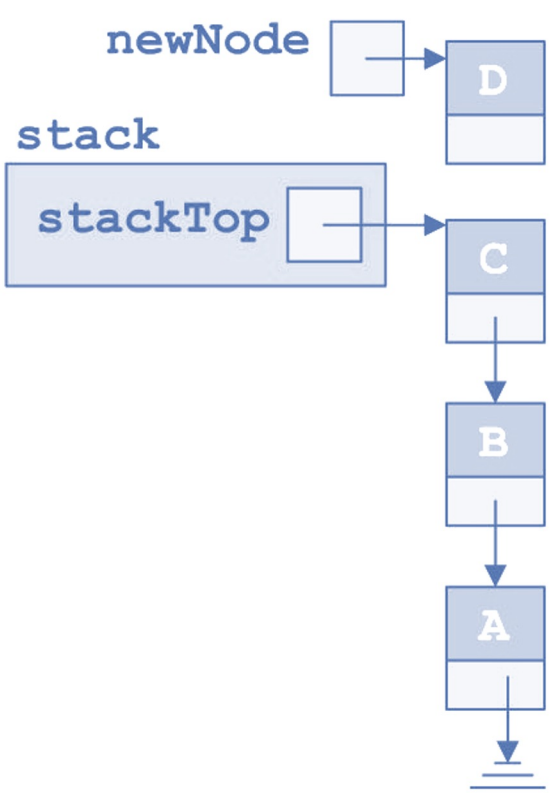
(b) Nonempty stack

I – Stacks – Linked List Implementation

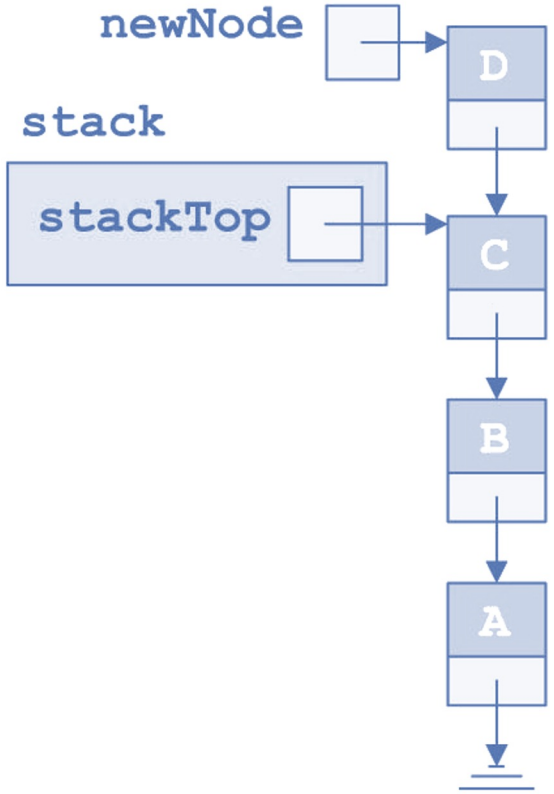
- Thêm phần tử vào đầu DSLK hay đỉnh của ngăn xếp

```
void push(int val)
{
    struct Node* newnode;
    newnode->data = val;
    newnode->next = top;
    top = newnode;
}
```

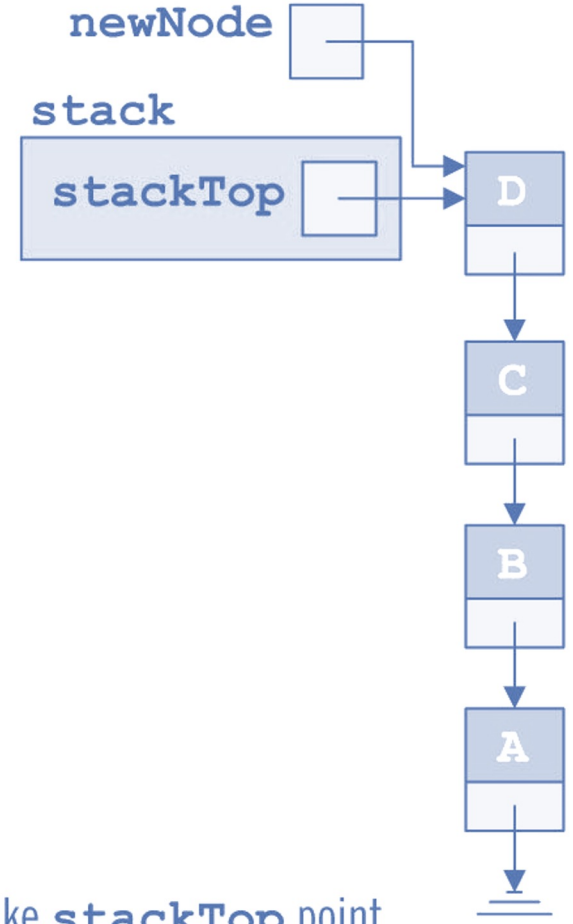
I – Stacks – Linked List Implementation



(a) Create **newNode** and store **D**



(b) Put **newNode** on the top of stack



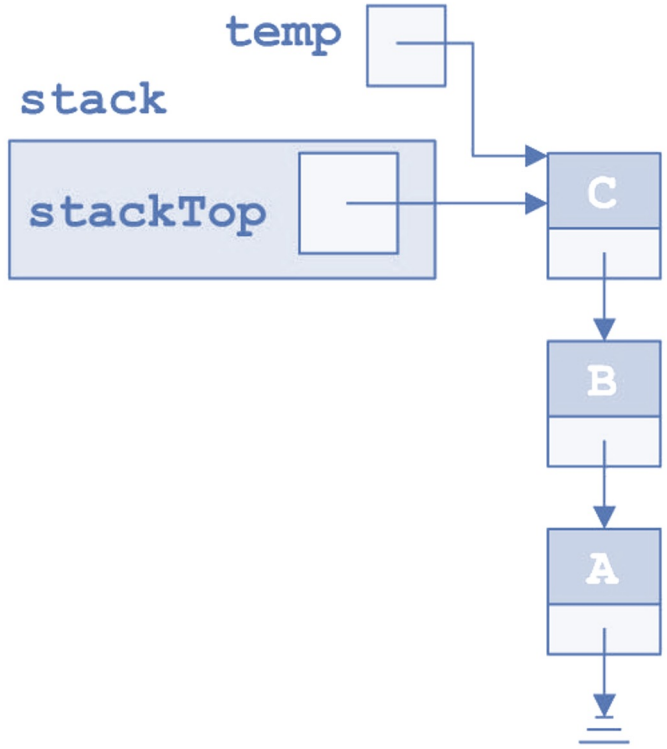
(c) Make **stackTop** point to the top element

I – Stacks – Linked List Implementation

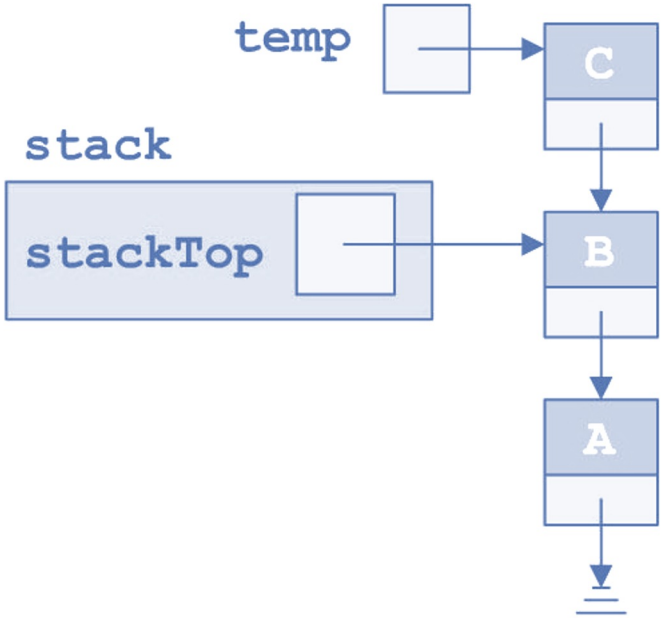
- Xóa phần phần tử đầu tiên của DSLK hay phần tử tại đỉnh của ngăn xếp

```
void pop()
{
    if(top==NULL)
        cout<<"Stack Underflow"<<endl;
    else
    {
        cout<<"The popped element is "<< top->data <<endl;
        top = top->next;
    }
}
```

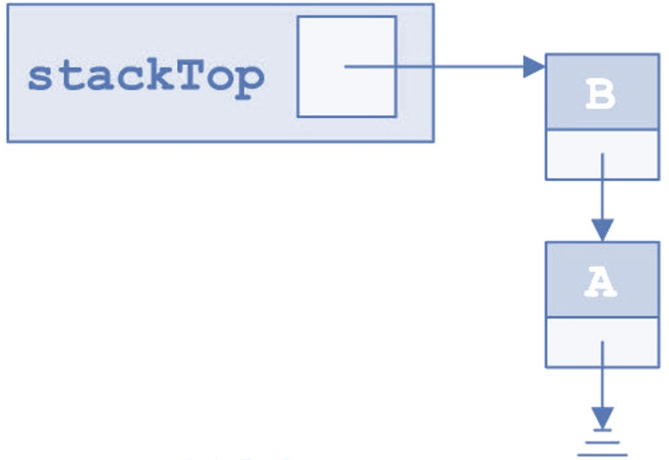
I – Stacks – Linked List Implementation



(a) Make **temp** point to the top element



(b) Make **stackTop** point to the next element



(c) Delete **temp**

I – Stacks – Linked List Implementation

- Hiển thị từng phần tử của ngăn xếp

```
void display()
{
    struct Node* current;
    if(top==NULL)
        cout<<"Stack is empty";
    else
    {
        current = top;
        cout<<"Stack elements are: ";
        while (current != NULL)
        {
            cout<< current->data <<" ";
            current = current->next;
        }
    }
    cout<<endl;
}
```

I - Stacks - STL C++

- Để sử dụng stack trong thư viện mẫu chuẩn của C++, ta sử thêm header như sau: #include <stack>
- Các hàm trong thao tác trên stack được hỗ trợ là

✓ `empty() – trả về true nếu ngăn xếp rỗng`

✓ `size() – trả về kích thước của ngăn xếp`

✓ `top() – trả về tham chiếu của phần tử ở đỉnh ngăn xếp`

✓ `push(g) – thêm phần tử g vào đỉnh của ngăn xếp`

✓ `pop() – xóa phần tử ở đỉnh của ngăn xếp`

I - Stacks - STL C++

```cpp
#include <iostream>
#include <stack>
using namespace std;
int main(){
    stack<int> stack; // khởi tạo stack
    stack.push(1); // thêm phần tử vào đỉnh của stack
    stack.push(3);
    stack.push(5);
    stack.push(6);

    stack.pop(); // xóa phần tử khỏi đỉnh của stack
    stack.pop();

    // in ra các phần tử của stack từ đỉnh xuống đáy
    while (!stack.empty()){
        cout << ' ' << stack.top();
        stack.pop();
    }
}
```

I – Stacks – Application – Prefix, Infix and Postfix

Ký hiệu trung tố (**infix**): ký hiệu thông thường để viết biểu thức số học

Toán tử được viết giữa các toán hạng

- Ví dụ: a + b
- Đánh giá từ trái sang phải
- Các toán tử có độ ưu tiên khác nhau
- Dấu ngoặc đơn () có thể được sử dụng để ghi đè mức độ ưu tiên

I – Stacks – Application – Prefix, Infix and Postfix

Biểu thức trung tố (**infix**): biểu thức thông thường để viết biểu thức số học

- Toán tử được viết giữa các toán hạng

  Ví dụ: a + b

- Đánh giá từ trái sang phải
- Các toán tử có độ ưu tiên khác nhau
- Dấu ngoặc đơn () có thể được sử dụng để ghi đè mức độ ưu tiên

I – Stacks – Application – Prefix, Infix and Postfix

Biểu thức tiền tố (**prefix**): toán tử được viết trước toán hạng

- Được giới thiệu bởi nhà toán học người Ba Lan Jan Lukasiewicz vào đầu những năm 1920
- Dấu ngoặc đơn có thể được bỏ qua
- Ví dụ: + a b

I – Stacks – Application – Prefix, Infix and Postfix

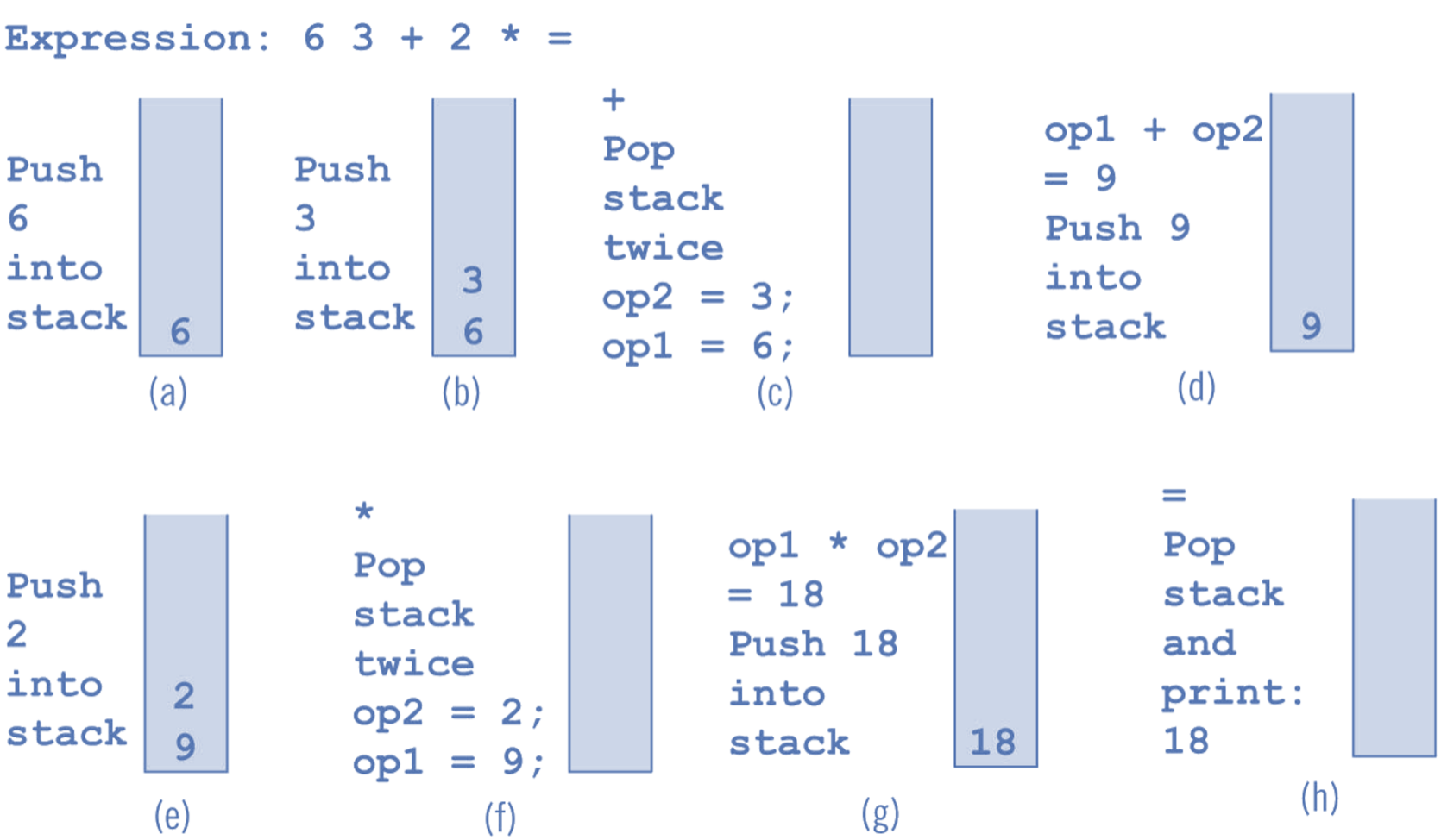
Biểu thức hậu tố (**postfix**) hay còn được gọi là ký pháp nghịch đảo ba lan (Reversed Polish notation):

- Được phát minh vào khoảng giữa thập kỷ 1950 bởi một triết học gia và nhà khoa học máy tính Charles Hamblin người Úc
- Nhiều trình biên dịch chuyển đổi biểu thức đại số sang biểu thức hậu tố trước khi chuyển sang ngôn ngữ máy
- Ưu điểm: các toán tử xuất hiện theo thứ tự cần thiết để tính toán
- Ví dụ: a + b * c = a b c * +
- Biểu thức được đọc từ trái sang phải, khi toán tử được xác định, thực hiện việc tính toán với hai toán hạng trước nó.

I – Stacks – Application – Prefix, Infix and Postfix

Infix Expression	Equivalent Postfix Expression
$a + b$	$a\ b\ +$
$a + b * c$	$a\ b\ c\ *\ +$
$a * b + c$	$a\ b\ *\ c\ +$
$(a + b) * c$	$a\ b\ +\ c\ *$
$(a - b) * (c + d)$	$a\ b\ -\ c\ d\ +\ *$
$(a + b) * (c - d / e) + f$	$a\ b\ +\ c\ d\ e\ /\ -\ *\ f\ +$

I – Stacks – Application – Prefix, Infix and Postfix



I – Stacks – Application – Prefix, Infix and Postfix

- +, -, *, và / là các toán tử yêu cầu hai toán hạng
  - Ngăn xếp thực hiện hàm pop() hai lần và thực hiện việc tính toán
  - Nếu ngăn xếp có ít hơn hai phần tử ---> lỗi
- Nếu toán tử là =, biểu thức kết thúc
  - Gọi hàm pop() để lấy kết quả từ ngăn xếp
  - Nếu ngăn xếp có nhiều hơn một phần tử ---> lỗi
- Nếu toán tử là bất cứ ký hiệu khác
  - Biểu thức chứa một toán tử không hợp lệ

Bài tập

1. Xác định thông tin được in ra màn hình của đoạn code sau

```
stack<int> stack;
int temp;
stack.push(28);
stack.push(16);
temp = stack.top();
stack.push(temp - 3);
cout << stack.top() << endl;
stack.push(2 * temp);
stack.push(50);
temp = stack.top()/3;
stack.pop();
stack.push(32);
while (!stack.empty())
{
    cout << stack.top() << " ";
    stack.pop();
}
cout << endl;
cout << "temp = " << temp << endl;
```

Bài tập

2. Thực hiện phép toán với các biểu thức hậu tố sau

   a. 12 18 – 30 + 2 * 3 / =

   b. 28 5 / 3 + 5 4 * – 8 + =

   c. 13 18 10 17 2 20 + – + /* =

3. Đổi các biểu thức trung tố sau sang biểu thức hậu tố

   a. x +y+z-w/t

   b. x – (y + z) * w + (t – u) /s

   c. ((x + y) / z * (u – v)) / t + w

II - Queues

- Hàng đợi (queue) là một tập các phần tử có cùng kiểu dữ liệu
- Các phần tử được thêm vào ở cuối và được xóa ở đầu hàng.
- Đây là đặc trưng của cấu trúc vào trước ra trước (first-in first-out, FIFO)
- Phần tử ở giữa hàng đợi không thể được truy cập

II - Queues

- Các thao tác trên hàng đợi
  - initializeQueue: khởi tạo hàng đợi
  - isEmpty: kiểm tra xem hàng đợi có rỗng hay không
  - isFull: kiểm tra xem hàng đợi đã đầy chưa
  - Front: lấy giá trị của phần tử ở đầu (front) của hàng đợi
  - Rear: lấy giá trị của phần tử ở cuối (back/rear) của hàng đợi
  - Enqueue: thêm phần tử vào cuối hàng đợi
  - Dequeue: xóa phần tử ở đầu hàng đợi

II - Queues

- Hàng đợi có thể được tạo và triển khai dựa trên một trong 3 cách sau

1. Sử dụng mảng 1 chiều (class)
2. Sử dụng DSLK (struct)
3. Sử dụng thư viện mẫu chuẩn (standard template library) C++

II – Queues – Array Implementation

- Biểu diễn hàng đợi dưới dạng mảng 1 chiều

```
#include <iostream>
using namespace std;
#define SIZE 100

class Queue
{
private:
        int array[SIZE];
        int front, rear;
public:
        Queue()
        bool isEmpty()
        bool isFull()
        void Enqueue(int x)
        void Dequeue()
        int Front()
        int Rear()
        void printQueue()
};
```

II - Queues - Array Implementation

- Khởi tạo hàng đợi

```
Queue()
{
        front = -1;
        rear = -1;
}
```

- Kiểm tra hàng đợi có rỗng hay không

```
bool IsEmpty()
{
        return (front == -1 && rear == -1);
}
```

II - Queues - Array Implementation

- Kiểm tra hàng đợi có đầy hay chưa

```
bool IsFull()
{
	return rear == SIZE - 1;
}
```

- Lấy giá trị của phần tử đầu tiên hàng đợi

```
int Front()
{
	if(front == -1)
	{
		cout<<"Error: cannot return front from
empty queue\n";
		return -1;
	}
	return array[front];
}
```

II - Queues - Array Implementation

- Lấy giá trị cuối cùng của hàng đợi

```
int Rear()
{
        if(rear == -1)
        {
                cout<<"Error: cannot return front from empty queue\n";
                return -1;
        }
        return array[rear];
}
```

II - Queues - Array Implementation

- Thêm phần tử vào cuối hàng đợi

```
void Enqueue(int x)
{
        cout<<"Enqueuing "<<x<<" \n";
        if(isFull())
        {
                cout<<"Error: Queue is Full\n";
                return;
        }
        if (isEmpty())
        {
                front = rear = 0;
        }
        else
        {
            rear = rear + 1;
        }
        array[rear] = x;
}
```

II - Queues - Array Implementation

- Xóa phần tử ở đầu hàng đợi

```
void Dequeue()
{
        cout<<"Dequeuing \n";
        if(isEmpty())
        {
                cout<<"Error: Queue is Empty\n";
                return;
        }
        else if(front == rear )
        {
                rear = front = -1;
        }
        else
        {
                front = front + 1;
        }
}
```

II - Queues - Array Implementation

- In các phần tử trong hàng đợi

```
void printQueue()
{
	cout<<"Queue: ";
	for(int i = front; i <= rear; i++)
	{
		cout<<array[i]<<" ";
	}
	cout<<"\n\n";
}
```

II – Queues – Linked List Implementation

- In các phần tử trong hàng đợi

```
void printQueue()
{
	cout<<"Queue: ";
	for(int i = front; i <= rear; i++)
	{
		cout<<array[i]<<" ";
	}
	cout<<"\n\n";
}
```

II – Queues – Linked List Implementation

- Mảng có kích thước cố định và cần phải được xử lý đặc biệt.
- Sử dụng DSLK sẽ giải quyết được vấn đề cố định.
- Phần tử mới sẽ được thêm vào ở cuối DSLK và phần tử ở đầu DSLK sẽ được loại bỏ. → cần 2 con trỏ
- Hàng đợi sẽ rỗng nếu con trỏ `front` trỏ tới `nullptr`
- Hàng đợi sẽ không đầy trừ khi `runs out of memory`

II - Queues - Linked List Implementation

- Các thao tác trên hàng đợi
  - initializeQueue: khởi tạo hàng đợi
  - isEmpty: kiểm tra xem hàng đợi có rỗng hay không
  - ~~isFull: kiểm tra xem hàng đợi đã đầy chưa~~
  - Front: lấy giá trị của phần tử ở đầu (front) của hàng đợi
  - Rear: lấy giá trị của phần tử ở cuối (back/rear) của hàng đợi
  - Enqueue: thêm phần tử vào cuối hàng đợi
  - Dequeue: xóa phần tử ở đầu hàng đợi

II – Queues – Linked List Implementation

- Khởi tạo hàng đợi

```
using namespace std;
struct node {
    int data;
    struct node * next;
};
struct node * front = nullptr;
struct node * rear = nullptr;
struct node * current;
```

II – Queues – Linked List Implementation

- Thêm phần tử vào cuối hàng đợi

```
void Enqueue(int val) {
    if (rear == nullptr) {
        rear = new node;
        rear -> next = nullptr;
        rear -> data = val;
        front = rear;
    } else {
        current = new node;
        rear -> next = current;
        current -> data = val;
        current -> next = nullptr;
        rear = current;
    }
}
```

II – Queues – Linked List Implementation

- Xóa phần tử đầu tiên của hàng đợi

```
void Dequeue() {
    current = front;
    if (front == nullptr) {
        cout << "Queue is empty!!" << endl;
    } else if (current -> next != nullptr) {
        current = current -> next;
        cout << "Element deleted from queue is : " << front -> data << endl;
        delete front;
        front = current;
    } else {
        cout << "Element deleted from queue is : " << front -> data << endl;
        delete front;
        front = nullptr;
        rear = nullptr;
    }
}
```

II - Queues - STL C++

- Để sử dụng queue trong thư viện mẫu chuẩn của C++, ta sử thêm header như sau: `#include <queue>`
- Các hàm trong thao tác trên `queue` được hỗ trợ là

✓ `queue <type> ourQueue;` khởi tạo queue với kiểu dữ liệu

✓ `empty()` – trả về true nếu hàng đợi rỗng

✓ `size()` – trả về kích thước của hàng đợi

✓ `front()` – trả về giá trị của phần tử ở đầu hàng đợi

✓ `back()` – trả về giá trị của phần tử ở cuối hàng đợi

✓ `push(x)` – thêm x vào cuối hàng đợi

✓ `pop()` – xóa phần tử ở đầu hàng đợi

II - Queues - STL C++

```
#include <iostream>
#include <queue>
using namespace std;
int main(void) {
    queue < int > myQueue;
    myQueue.push(3);
    myQueue.push(6);
    myQueue.push(7);

    cout << "The queue size is = " << myQueue.size() << endl;

    while (!myQueue.empty()) {
        cout << myQueue.front() << endl;
        myQueue.pop();
    }

    return 0;
}
```

II - Queues - Application - Simulation

- Mô phỏng (simulation) là kỹ thuật mà máy tính sử dụng để mô hình hóa các vấn đề thực tế
- Server: đối tượng cung cấp dịch vụ
- Customer: đối tượng sử dụng dịch vụ
- Transaction time: thời gian sử dụng dịch vụ của 1 khách hàng.
- Model: hệ thống bao gồm các server và hàng đợi chứa các thứ tự các khách hàng. Khách hàng đứng đầu tiên trong hàng đợi sẽ là người được sử dụng server tiếp theo.

II – Queues – Application - Simulation

- Các thông tin cần biết
  - Số lượng server
  - Thời gian khách hàng bắt đầu sử dụng server
  - Thời gian giữa các khách hàng
  - Các sự kiện ảnh hưởng tới hiệu suất của hệ thống
  - ...
- Hiệu suất của hệ thống phụ thuộc vào các yếu tố
  - Số lượng server
  - Thời gian một khác hàng sử dụng server
  - Tần suất sử dụng server của khách hàng
  - ...

Bài tập

- Bài 1: Xác định thông tin được in lên màn hình của đoạn code sau

```
queue < int > myQueue;
int temp = 20;

myQueue.push(15);
myQueue.push(8);
myQueue.push(temp);
myQueue.push(42);
temp = myQueue.front();
myQueue.pop();
myQueue.push(temp - 10);
myQueue.push(16);
myQueue.pop();
cout << "Queue elements: ";
while (!myQueue.empty())
{
    cout << myQueue.front() << " ";
    myQueue.pop();
}
cout << endl;
cout << "temp = " << temp << endl;
```

Bài tập

- Bài 2: Biết hàng đợi được biểu diễn bằng mảng, viết định nghĩa hàm moveNthFront nhận vào một số nguyên dương n và di chuyển phần tử thứ n trong hàng đợi lên vị trí đầu tiên. Thứ tự của các phần tử khác giữ nguyên không đổi.
  a) Sử dụng thêm hàng đợi để giải quyết bài toán trên
  b) Sử dụng thêm ngăn xếp để giải quyết bài toán trên

Ví dụ

- Input: queue = {5, 11, 34, 67, 43, 55} and n = 3.
- Output: queue = {34, 5, 11, 67, 43, 55}

Bài tập

- Bài 3: Cho hàng đợi A. Trình bày ý tưởng để đảo ngược các phần tử trong hàng đợi A
  a) Chỉ sử dụng hàng đợi
  b) Chỉ sử dụng ngăn xếp
  c) Vừa sửa dụng hàng đợi và ngăn xếp
- Bài 4: Cho ngăn xếp B. Trình bày ý tưởng để đảo ngược các phần tử trong ngăn xếp B
  a) Chỉ sử dụng hàng đợi
  b) Chỉ sử dụng ngăn xếp
  c) Vừa sửa dụng hàng đợi và ngăn xếp

THANK YOU FOR YOUR ATTENTIONS