# Data Structure & Algorithm

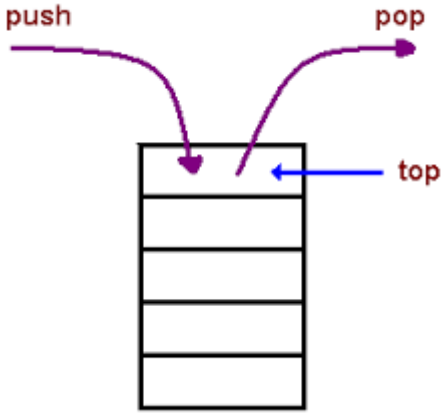
# Topic 4: Các kiểu dữ liệu trừu tượng cơ bản- Ngăn xếp (stack)

# Ngăn xếp – stack

- Khái niệm Ngăn xếp
- Các phép toán trên Ngăn xếp
- Cài đặt ngăn xếp bằng DSLK
- Cài đặt ngăn xếp bằng mảng

# Khái niệm Ngăn xếp - Stack

- **Stack**
  - Là một danh sách mà ta giới hạn việc thêm vào hoặc loại bỏ phần tử chỉ thực hiện tại một đầu của danh sách (đỉnh của danh sách)
  - Nguyên tắc hoạt động: LIFO (*Last in – first out*): vào sau ra trước
  - Stack rỗng là stack không chứa phần tử nào



# Ngăn xếp (Stack)

- Stack
  - gồm nhiều phần tử có cùng kiểu dữ liệu
  - Phần tử trên cùng stack luôn có một con trỏ chỉ tới, gọi là Stack Pointer (sp)
- Có 2 cách cài đặt
  - Mảng
  - Danh sách liên kết

# Ngăn xếp (Stack)

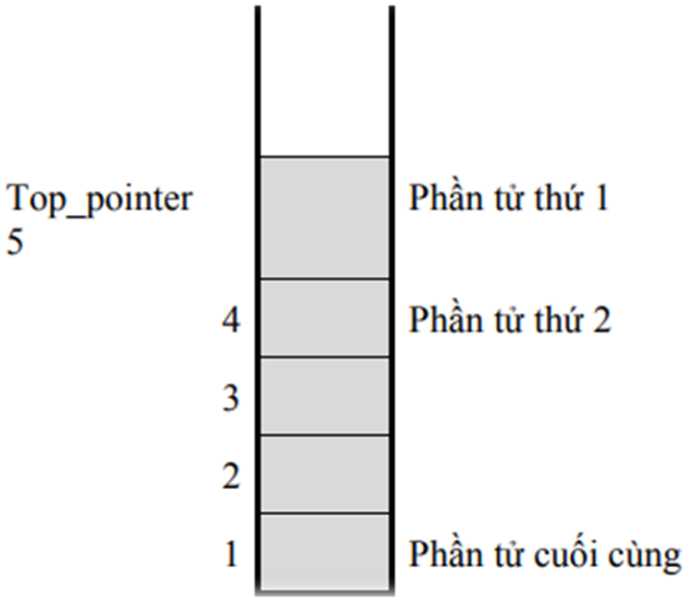
- Các thao tác cơ bản
  - Tạo ngăn xếp rỗng: MakeNull_Stack
  - Lấy phần tử tại đỉnh của Stack: Top
  - Xóa phần tử tại đỉnh của Stack: Pop
  - Thêm phần tử vào đỉnh của Stack: Push
  - Kiểm tra Stack có rỗng không: isEmpty_Stack

# Cài đặt Stack

- Cài đặt bằng Danh sách
- Cài đặt bằng Mảng

# Cài đặt Stack bằng Mảng

- Dùng 1 mảng để lưu trữ liên tiếp các phần tử của Stack.
- Các phần tử đưa vào Stack bắt đầu từ vị trí có chỉ số thấp nhất của mảng.
- Dùng 1 biến số nguyên Top_Pointer giữ lại vị trí của phần tử đỉnh.
- Giả sử vùng nhớ tối đa được dùng cho Stack là MaxLength



# Cài đặt Stack bằng Mảng

- **Khai báo**

```
#define MaxLength 100 //Giả sử Stack có tối đa 100 phần tử
typedef int ElementType
typedef struct Stack
{
    int Top_Pointer; //Lưu lại vị trí của phần tử đỉnh
    ElementType Elements[MaxLength]; //Mảng chứa giá trị các
                                     //phần tử
};
```

# Cài đặt Stack bằng Mảng

- Cài đặt thao tác
  - **1. Tạo Stack rỗng** MakeNull_Stack

    Stack rỗng: Top_Pointer = 0

```
void MakeNull_Stack (Stack &st)
{
        st.Top_Pointer = 0;
}
```

# Cài đặt Stack bằng Mảng

- Cài đặt thao tác
  - **2. Kiểm tra Stack có rỗng không**: isEmpty_Stack

    Trả về TRUE nếu rỗng, ngược lại thì trả về FALSE

```
int isEmpty_Stack (Stack st)
{
        return(st.Top_Pointer == 0);
}
```

# Cài đặt Stack bằng Mảng

- Cài đặt thao tác
  - **3. Lấy phần tử tại đỉnh của Stack**: Top
    - Nếu Stack rỗng thì không thông báo.

```
ElementType Top (Stack st)
{
    ElementType top;
    if (isEmpty_Stack(st))
        printf ("Stack rong!\n");
    else
        top = st.Elements[st.Top_Pointer-1];
    return top;
}
```

# Cài đặt Stack bằng Mảng

- Cài đặt thao tác
  - **4.** Xóa phần tử tại đỉnh của Stack: Pop
    - Nếu Stack rỗng thì không thực hiện

```
void Pop (Stack &st)
{
    if (isEmpty_Stack(st))
        printf ("Stack rong. Khong the xoa!\n");
    else
        st.Top_Pointer --;

}
```

# Cài đặt Stack bằng Mảng

- Cài đặt thao tác
  - **5.** Thêm phần tử vào đỉnh của Stack: Push
    - Nếu Stack đầy thì không thực hiện

```
void Push (Stack &st, ElementType X)
{
    if (st.Top_Pointer == MaxLength ) printf ("Stack
    day!\n");
    else
    {
        st.Elements[st.Top_Pointer] = X;
        st.Top_Pointer ++;
    }
}
```

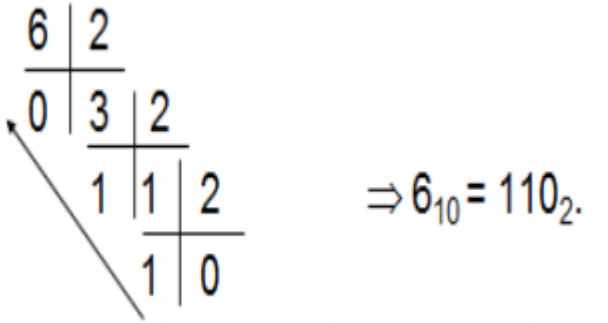
# Cài đặt Stack bằng Mảng

- Ứng dụng
  - Stack thường được dùng trong các bài toán có cơ chế LIFO
  - Stack cũng được dùng trong các bào toán gỡ đệ qui (chuyển một giải thuật đệ quy thành giải thuật KHÔNG đệ quy)

# Cài đặt Stack bằng Mảng

- **Ví dụ**: Viết chương trình đổi số nguyên không âm ở hệ thập phân sang số ở hệ nhị phân.

1. Các số dư được tạo ra sau lại được hiển thị trước. Cơ chế sắp xếp này phù hợp với cơ chế hoạt động của stack.
2. Dùng 1 stack để lưu trữ các số dư, sau đó lại lấy các số này từ stack ra để hiển thị thành mã số nhị phân.



# Cài đặt Stack

- Cài đặt bằng Danh sách
- Cài đặt bằng Mảng

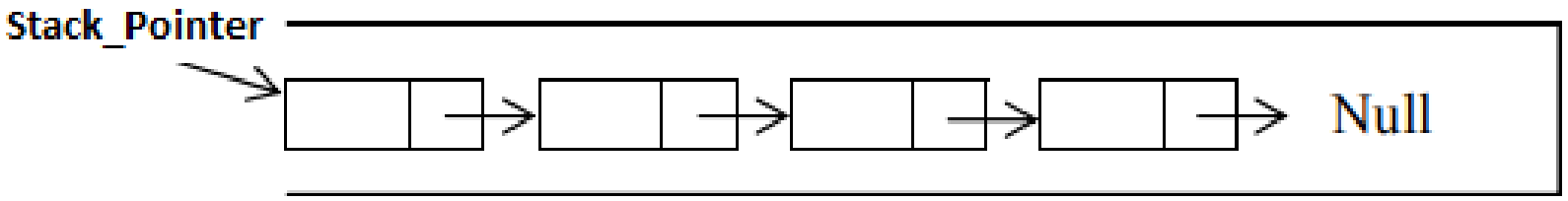
# Cài đặt bằng Danh sách liên kết

- **Khai báo**
  - Khai báo biến **ST** (*Stack Pointer*): con trỏ chỉ đến 1 danh sách là Stack
  - Mỗi phần tử trong Stack là 1 node như sau:

```
typedef int ElementType
typedef struct Node
{
    ElementType Data; //Lưu dữ liệu
    Node * Next;
};
typedef Node *Stack;
```

# Cài đặt bằng Danh sách liên kết

- **Khai báo**
  - **Lưu ý:**
    - Với Stack là DSLK, ta chỉ thực hiện các phép toán ở đầu danh sách.
    - KHÔNG có trường hợp Stack đầy
    - Stack rỗng khi ST = NULL.



# Cài đặt bằng Stack bằng DSLK

- Cài đặt thao tác
  - **1. Tạo Stack rỗng** MakeNull_Stack

    Stack rỗng: ST = NULL

```
void MakeNull_Stack (Stack &st)
{
        st = NULL;
}
```

# Cài đặt bằng Stack bằng DSLK

- Cài đặt thao tác
  - **2. Kiểm tra Stack có rỗng không**: isEmpty_Stack

    Trả về TRUE nếu rỗng, ngược lại thì trả về FALSE

```
int isEmpty_Stack (Stack st)
{
        return (st ==  NULL);
}
```

# Cài đặt bằng Stack bằng DSLK

- Cài đặt thao tác
  - **3. Lấy phần tử tại đỉnh của Stack**: Top
    - Nếu Stack rỗng thì không thông báo.

```
ElementType Top (Stack st)
{
    ElementType top;

    if (isEmpty_Stack)
        printf("Stack rong! Khong co phan tu dinh\n");
    else top = st->Data;

    return top;
}
```

# Cài đặt bằng Stack bằng DSLK

- Cài đặt thao tác
  - **4. Xóa phần tử tại đỉnh của Stack:** Pop
    - Nếu Stack rỗng thì không thực hiện

```
void Pop (Stack &st)
{
    if(isEmpty_Stack(st))
        printf("Stack rong. Khong xoa duoc!\n");
    else
    {
        Node *temp = st;
        st = temp->Next;
        free (temp);
    }
}
```

# Cài đặt bằng Stack bằng DSLK

- Cài đặt thao tác
  - **5. Thêm phần tử vào đỉnh của Stack**: Push

```
void Push (Stack &st, ElementType X)
{
    Node * temp = CreateNewNode(X);
    if(!isEmpty_Stack(st)) temp->Next = st;
    st = temp;
}
```