# *Cấu trúc dữ liệu và Thuật toán*

## Danh sách liên kết
## Stacks

## *Giới hạn của mảng*

- **Mảng (Arrays)**
  - **Đơn giản,**
  - **Nhanh**

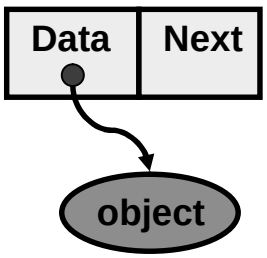
  ***nhưng***
  - **Phải chỉ định một kích thước cụ thể tại thời điểm xây dựng mảng**
  - **Tuân thủ Luật Đầy (Murphy)**
    - **Xây dựng một mảng với không gian cho *n* biến**
      - ***n* = Ước lượng chính chắn của bạn về số lượng biến sẽ sử dụng lớn nhất**
    - **Ngày mai, bạn có thể cần n+1 biến**
  - **Liệu có thể có một hệ thống mềm dẻo hơn?**

## *Danh sách liên kết*

- Sử dụng không gian biến mềm dẻo
  - Cấp phát không gian động cho từng phần tử theo yêu cầu của bài toán
  - Gắn một con trỏ chỉ tới giá trị phần tử  kế tiếp

## ←Danh sách liên kết

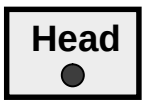
- Mỗi nút (node) của danh sách bao gồm
  - Giá trị khóa của phần tử trong nút
  - Con trỏ chỉ đến node kế tiếp



## *Danh sách liên kết*

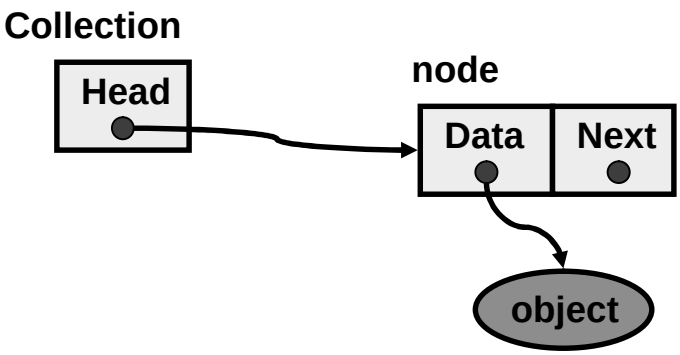
- **Cấu trúc danh sách liên kết chứa một con trỏ tai đầu danh sách: head**
  - **Nó được khởi tạo = NULL**

**Danh sách liên kết**



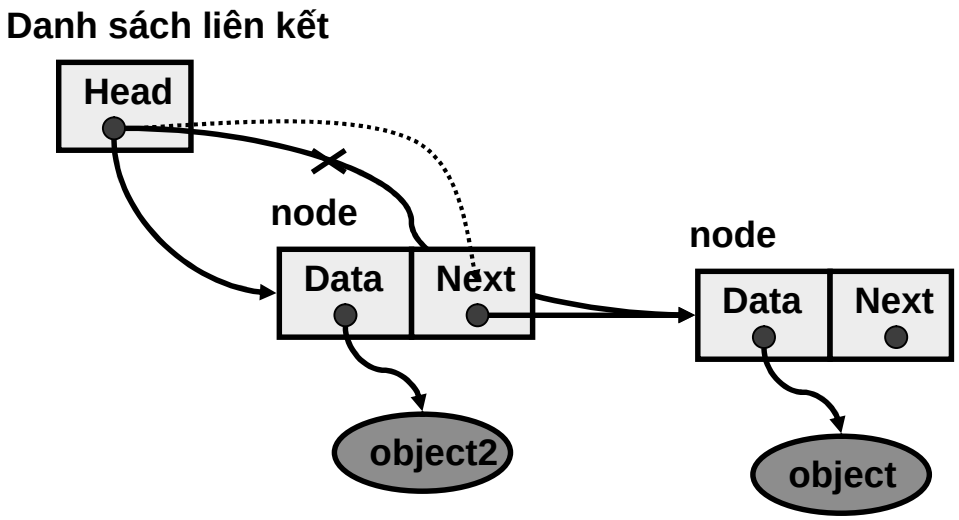
## *Danh sách liên kết*

- **Cấu trúc danh sách liên kết chứa một con trỏ tai đầu danh sách: head**
  - **Nó được khởi tạo = NULL**
- **Bổ xung phần tử đầu tiên**
  - **Cấp phát không gian vùng nhớ cho node**
  - **Thiết lập giá trị dữ liệu ứng với đối tượng cần mô tả**
  - **Thiết lập Next đến NULL**
  - **Thiết lập Head để trỏ đến node mới**



# *Danh sách liên kết*

- **Bổ xung phần tử thứ hai**
  - **Cấp phát không gian vùng nhớ cho node**
  - **Thiết lập giá trị dữ liệu ứng với đối tượng cần mô tả**
  - **Thiết lập Next trỏ đến vị trí Head hiện tại trỏ đến**
  - **Thiết lập Head trỏ đến node mới**



## *Danh sách liên kết - Thủ tục bổ xung* Add

- **Thủ tục bổ xung**

```
struct t_node {
    void *item;
    struct t_node *next;
    } node;
typedef struct t_node *Node;
struct collection {
    Node head;
    ……
    };
int AddToCollection( Collection c, void *item ) {
    Node new = malloc( sizeof( struct t_node ) );
    new->item = item;
    new->next = c->head;
    c->head = new;
    return TRUE;
    }
```

# *Danh sách liên kết - Thủ tục bổ xung* Add

- **Thủ tục bổ xung**

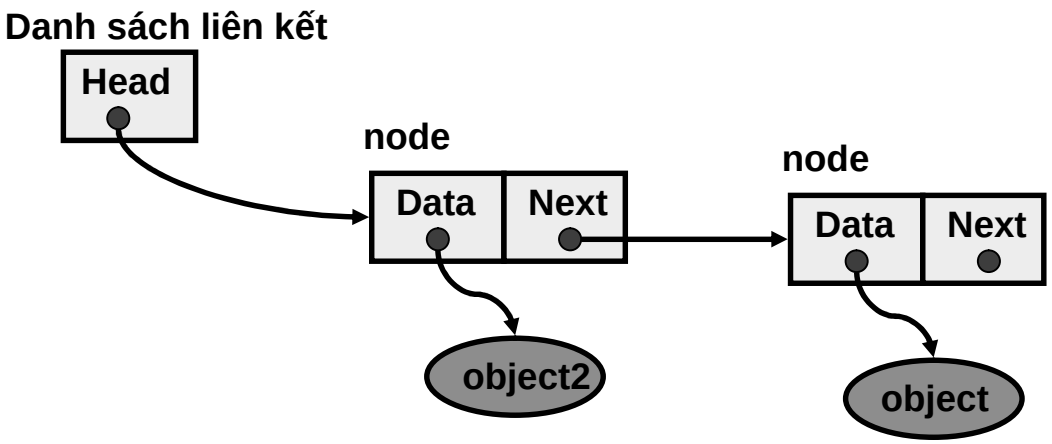
```
struct t_node {
    void *item;
    struct t_node *next;
    } node;
typedef struct t_node *Node;
struct collection {
    Node head;
    ……
    };
int AddToCollection( Collection c, void *item ) {
    Node new = malloc( sizeof( struct t_node ) );
    new->item = item;
    new->next = c->head;
    c->head = new;
    return TRUE;
    }
```

Định nghĩa kiểu đệ qui - C cấp phát vùng nhớ!

Kiểm tra lỗi, chèn thêm một dòng lệnh để k tra!

## *Danh sách liên kết*

- **Thời gian bổ xung**
  - **Hằng số - độc lập với n**
- **Thời gian tìm kiếm**
  - **Trường hợp tồi nhất - n**



## *Danh sách liên kết – Thủ tục* `Find`

- **Thủ tục**

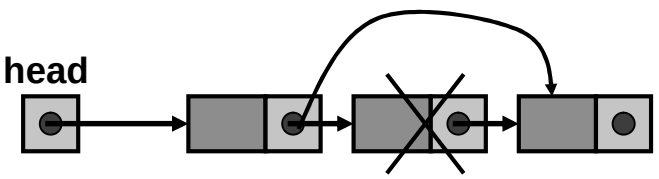
```
void *FindinCollection( Collection c, void *key ) {
    Node n = c->head;
    while ( n != NULL ) {
      if ( KeyCmp( ItemKey( n->item ), key ) == 0 ) {
            return n->item;
      n = n->next;
      }
    return NULL;
    }
```

- ***Một thủ tục đệ qui cũng có thể sử dụng!***

## *Danh sách liên kết – Thủ tục* `Delete`

- **Thủ tục**

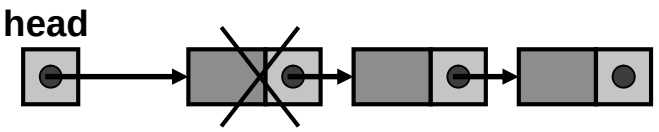
```
void *DeleteFromCollection( Collection c, void *key ) {
    Node n, prev;
    n = prev = c->head;
    while ( n != NULL ) {
      if ( KeyCmp( ItemKey( n->item ), key ) == 0 ) {
            prev->next = n->next;
            return n;
            }
      prev = n;
      n = n->next;
      }
    return NULL;
    }
```



## *Danh sách liên kết – Thủ tục* `Delete`

- **Thủ tục**

```
void *DeleteFromCollection( Collection c, void *key ) {
    Node n, prev;
    n = prev = c->head;
    while ( n != NULL ) {
      if ( KeyCmp( ItemKey( n->item ), key ) == 0 ) {
            prev->next = n->next;
            return n;
            }
      prev = n;
      n = n->next;
      }
    return NULL;
    }
```

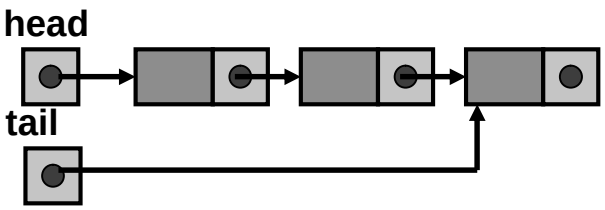



**Cần bổ xung thêm một số lệnh Để xóa phần tử! Bài tập!**

## *Danh sách liên kết - LIFO và FIFO*

- **Thủ tục**
  - **Bổ xung tại đầu danh sách (head)**

  **🡐Last-In-First-Out (LIFO) semantics**
- **Hiệu chỉnh**
  - **First-In-First-Out (FIFO)**
  - **Giữ con trỏ đuôi tail**




```
struct t_node {
    void *item;
    struct t_node *next;
    } node;
typedef struct t_node *Node;
struct collection {
    Node head, tail;
    };
```

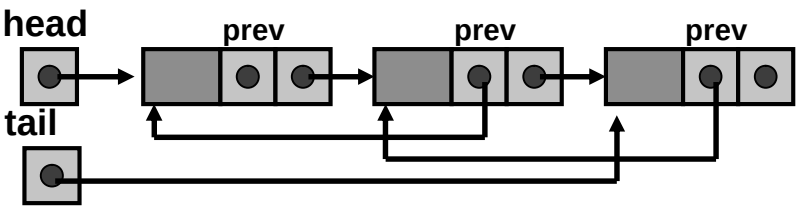
`tail` được thiết lập trong `AddToCollection` Nếu `head == NULL`

## *Danh sách liên kết – Liên kết kép*

- **Danh sách liên kết kép**
  - **Có thể trỏ cả hai hướng**

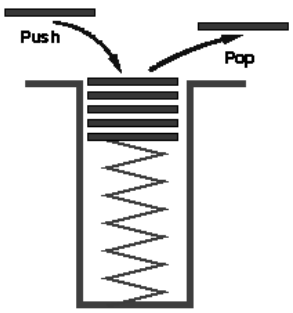
```
struct t_node {
    void *item;
    struct t_node *prev,
                  *next;
    } node;

typedef struct t_node *Node;
struct collection {
    Node head, tail;
    };
```



## *Stacks*

- **Stacks là một dạng danh sách (mảng) đặc biệt với ngữ cảnh LIFO**
- **Hai phương pháp**
  - `int push( Stack s, void *item );`
    **- Bổ xung một phần tử vào đỉnh của stack**
  - `void *pop( Stack s );`
    **- Loại bỏ một phần tử từ đỉnh của stack**
- **Tương tự như một máy xếp đĩa**
- **Các phương pháp khác**




```
int IsEmpty( Stack s );
/* Return TRUE if empty */
void *Top( Stack s );
/* Return the item at the top,
    without deleting it */
```

## *Stacks – Thủ tục*

- **Mảng (Arrays)**
  - **Cung cấp một stack với sức chứa giới hạn**
  - **Hạn chế về độ mềm dẻo *nhưng* đáp ứng được nhiều ứng dụng thực tế**
    - **Giới hạn về sức chứa bởi một vài ràng buộc**
      - **Bộ nhớ trong máy tính của bạn**
      - **Kích cỡ của máy xếp đĩa, etc**
- **Phương pháp `push`, `pop`**
  - **Các biến của `AddToC…`, `DeleteFromC…`**
- **Danh sách liên kết cũng được sử dụng**
- **Stack:**
  - ***Về cơ bản là một danh sách liên kết với ngữ nghĩa đặc biệt!***

## *Stacks – một số vấn đề liên quan*

- **Stacks trong chương trình tin học**
  - **Là khóa để call / return trong functions & procedures**
  - **Khuôn dạng Stack cho phép các lời gọi đệ qui**
  - **Call: push stack frame**
  - **Return: pop stack frame**
- **Khuôn dạng Stack**
  - **Các tham số của Function**
  - **Return địa chỉ**
  - **Các biến cục bộ (Local variables)**

## *Stacks – Thủ tục*

```
struct t_node {
    void *item;
    struct t_node *prev,
                  *next;
    } node;

typedef struct t_node *Node;
struct collection {
    Node head, tail;
    };
```

**prev không bắt buộc!**