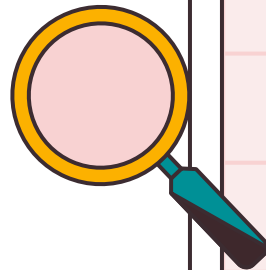
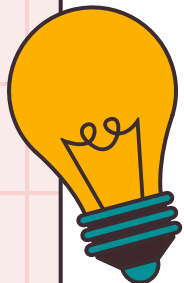




NHÓM 6

Câu Trắc Dữ Liệu Động

GVDH: Dương Việt Hằng





DANH SÁCH THÀNH VIÊN

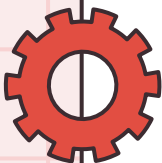
1. Trần Hoàng Anh Thư

2. Đỗ Trà Ngọc Châu

3. Trần Ngọc Cẩm Nguyên



NỘI DUNG



01

**KHÁI NIỆM VÀ VAI
TRÒ CỦA CSDL ĐỘNG**

04

**DANH SÁCH
LIÊN KẾT ĐƠN**

02

**Kiểu DỮ LIỆU
CON TRỎ**

05

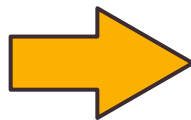
**TỔ CHỨC DANH SÁCH
LIÊN KẾT ĐƠN**

03

DANH SÁCH LIÊN KẾT

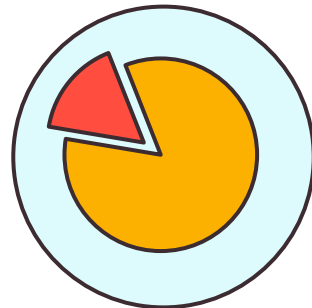
06

**CÁC THAO TÁC
TRÊN DANH SÁCH
LIÊN KẾT ĐƠN**

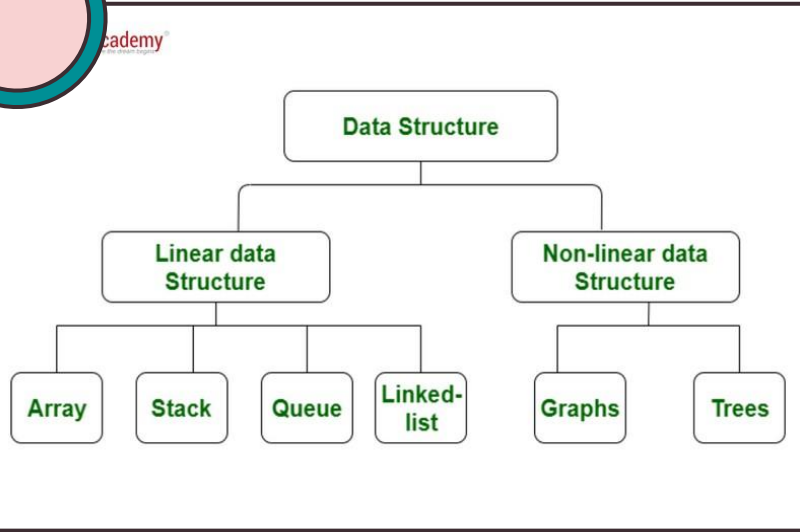


01

KHÁI NIỆM và vai TRÒ của CƠ SỞ DỮ LIỆU ĐỘNG



KHÁI NIỆM



- Cấu trúc dữ liệu động là những thứ có thể mở rộng hoặc thu lại tùy thuộc vào chương trình, đồng thời các vị trí bộ nhớ liên quan của chúng sẽ có thể thay đổi (vd: Danh sách liên kết được tạo ra bằng con trỏ)

Vai Trò

Quản lý bộ
nhớ động

Chuỗi

Danh sách
liên kết

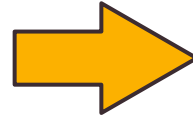
Hàng đợi và
ngăn xếp

Cây và đồ
thị

Chuỗi kí tự

Cấu Trúc
Dữ Liệu
Hàng Đợi
Ưu Tiên

⇒ Sử dụng cấu trúc dữ liệu động giúp giải quyết nhiều vấn đề liên quan đến quản lý và xử lý dữ liệu trong các ứng dụng lập trình thực tế.



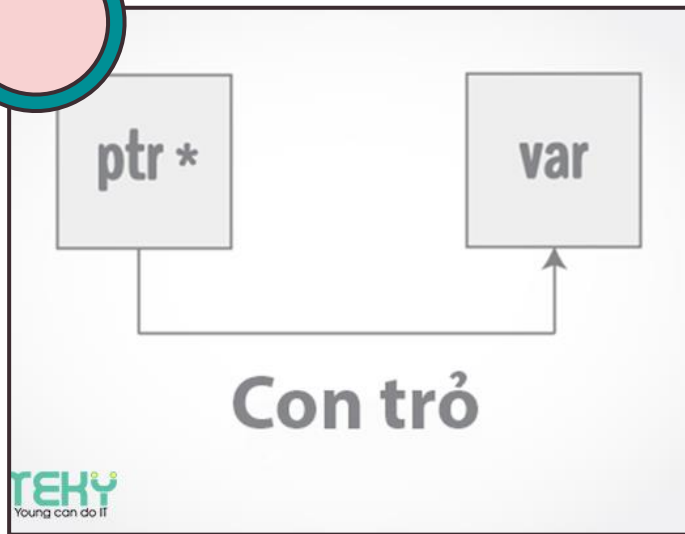
02

Kiểu Dữ Liệu CON TRỎ



KHái Niệm

- Con trỏ là một biến mà trong đó giá trị của nó là địa chỉ của biến khác.



Ví Dụ

Khởi tạo con trỏ

```
int *ptr; // Khai báo con trỏ kiểu int
int x = 10;
ptr = &x; // Gán địa chỉ của x cho con trỏ ptr
```

Truy cập dữ liệu

```
cout << "Giá trị của x: " << *ptr << endl;
*ptr = 20; // Thay đổi giá trị của x
           thông qua con trỏ
```

Con trỏ và mảng động

```
int arr = new int [100]; //Tạo mảng động
                           bằng con trỏ
```

Ví Dụ

Cấp phát động bộ nhớ

```
int *dynamicPtr = new int;  
*dynamicPtr = 30;
```

Giải phóng bộ nhớ

```
delete dynamicPtr; // Giải  
phóng bộ nhớ đã được cấp phát  
động
```

Con trỏ và hàm

```
void modifyValue(int  
*ptr) {  
    *ptr = 50;}  

```

LỢI ÍCH

Quản lý bộ
nhớ động

Quản lý file

Chuyển tham
chiếu trong hàm

Tạo ra cấu trúc
dữ liệu phức tạp

Thao tác
với mảng

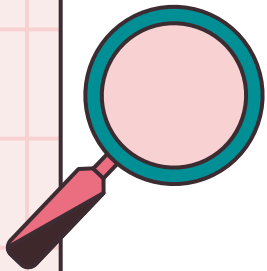
Hiệu quả và
tối ưu hoá

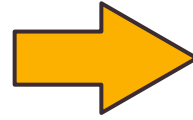
Cấu trúc dữ
liệu linh hoạt

Đa luồng

Điều hướng và
xử lý chuỗi

Thực hiện các
CTDL phức tạp



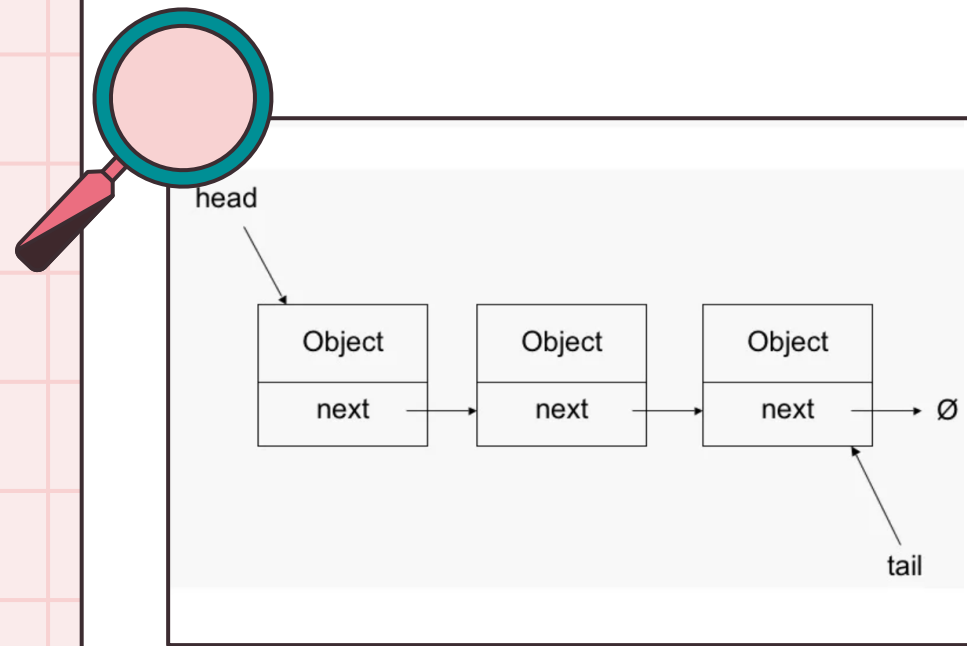


03

**DANH SÁCH
Liên kết**



KHÁI NIỆM



- Một Danh sách liên kết (Linked List) là một dãy các cấu trúc dữ liệu được kết nối với nhau thông qua các liên kết (link).
- Bao gồm một nhóm các nút (node) tạo thành một chuỗi.
- Danh sách liên kết là cấu trúc dữ liệu được sử dụng phổ biến thứ hai sau mảng.

CÁC KHÁI NIỆM CƠ BẢN

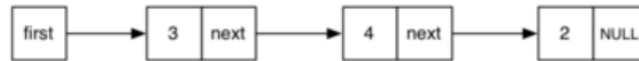
- *Link (liên kết)*: mỗi link của một Danh sách liên kết có thể lưu giữ một dữ liệu được gọi là một phần tử.
- *Next*: Mỗi liên kết của một Danh sách liên kết chứa một link tới next link được gọi là Next.
- *First*: một Danh sách liên kết bao gồm các link kết nối tới first link được gọi là First.
- Biểu diễn Danh sách liên kết (Linked List):



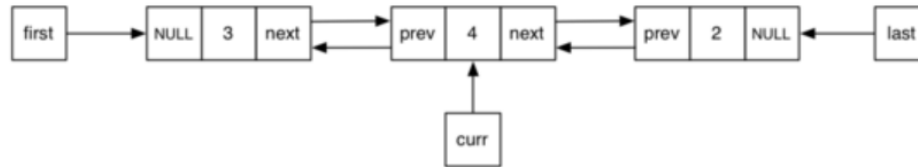
CÁC KIỂU TỔ CHỨC LIÊN KẾT

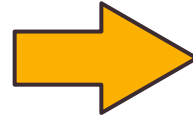
- Danh sách liên kết đơn (Simple Linked List)
- Danh sách liên kết đôi (Doubly Linked List)
- Danh sách liên kết vòng (Circular Linked List)
- ...

Singly-linked List



Doubly-linked List

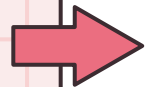




04

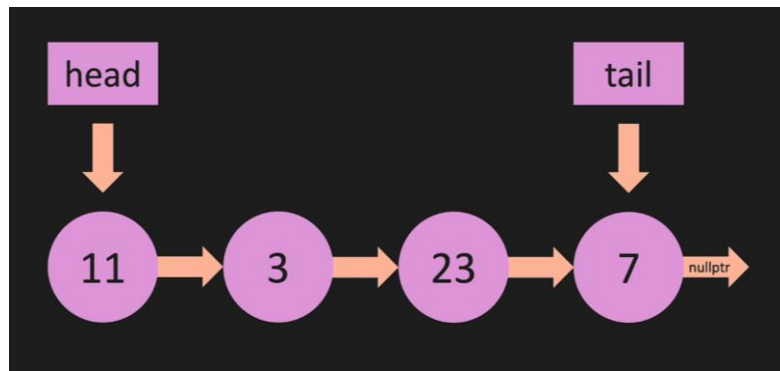
DANH SÁCH LIÊN KẾT ĐƠN

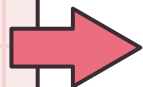




KHÁI NIỆM

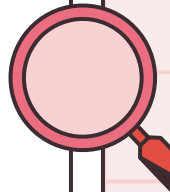
- Danh sách liên kết đơn là một loại danh sách liên kết đơn hướng, nghĩa là nó chỉ có thể được duyệt theo một hướng từ nút đầu (head) đến nút cuối cùng (tail), với mỗi phần tử chỉ liên kết với phần tử đứng sau nó trong danh sách.





Mỗi phần tử trong DSLK đơn là một cấu trúc có hai thành phần:

- Thành phần dữ liệu (Data): Lưu trữ thông tin.
- Thành phần liên kết (Next pointer) : Lưu địa chỉ phần tử đứng sau trong danh sách hoặc bằng NULL nếu là phần tử cuối danh sách.





Đặc Điểm




Các phần tử được lưu trữ ngẫu nhiên

Có thể thay đổi kích thước

Truy cập tới phần tử ngẫu nhiên
phải duyệt từ đầu đến vị trí đó

Chỉ có thể tìm kiếm tuyến tính

Kích thước tối đa phụ thuộc vào
bộ nhớ



Sử dụng tối ưu bộ nhớ

Bộ nhớ có thể bị phân mảnh sau một thời gian thực hiện các thao tác xin cấp phát và giải phóng.

Nếu chương trình xin cấp phát một vùng nhớ có kích thước m bytes nhưng không có vùng nhớ liên tục nào có kích thước lớn hơn hoặc bằng m bytes (cho dù tổng vùng nhớ còn trống thì đủ) thì hệ thống cũng không thể cấp phát được.



Vùng nhớ đang sử dụng

Vùng nhớ chưa sử dụng

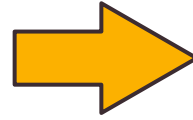
DSLK đơn có thể giải quyết



➔ Truy cập tuần tự

Để đến được 1 phần tử trong danh sách liên kết đơn, phải đi từ phần tử đầu tiên, lần theo các mối liên kết để đến được phần tử cần truy xuất. Chi phí để thực hiện công việc này là tuyến tính.


➔ Thao tác phức tạp




05

TỔ CHỨC DANH SÁCH LIÊN KẾT ĐƠN





```
1 struct Node
2 {
3     int data;
4     Node* next;
5 };
```



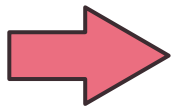
```
1 struct LinkedList
2 {
3     Node* Head;
4     Node* Tail;
5 };
```

Sử dụng struct để cài đặt Node và LinkedList






```
1 void Init(LinkedList& a)
2 {
3     a.Head = NULL;
4     a.Tail = NULL;
5 }
```

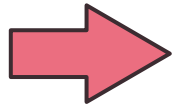


Tạo một DSLK đơn rỗng





```
1  Node* MakeNode(int x)
2  {
3      Node* newNode = new Node;
4      newNode->data = x;
5      newNode->next = NULL;
6      return newNode;
7  }
```

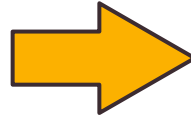


Tạo một Node mới





CÁC THAO TÁC TRÊN DANH SÁCH LIÊN KẾT ĐƠN



06





1

Tạo 1 danh sách liên kết đơn rỗng

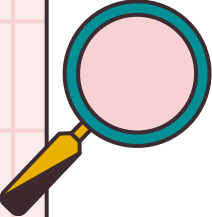
2

Tạo 1 nút có trường Infor bằng x

3

Tìm một phần tử có Info bằng x

1. Tạo 1 danh sách liên kết đơn



```
//Khởi tạo danh sách rỗng  
void Init(LinkedList &a)  
{  
    a.Head = NULL;  
    a.Tail = NULL;  
}
```

2. Tạo 1 nút có trường Infor bằng x

```
// Thêm một node vào vị trí được chọn trong danh sách
void InsertAt(LinkedList& a, int x, int position)
{
    if (position <= 0)
    {
        cout << "Vi tri chen khong hop le!" << endl;
        return;
    }

    if (position == 1)
    {
        Prepend(a, x);
        return;
    }
}
```



2. Tạo 1 nút có trường Infor bằng x

```
Node* current = a.Head;
int currentPosition = 1;
while (currentPosition < position - 1 && current != NULL)
{
    current = current->next;
    currentPosition++;
}

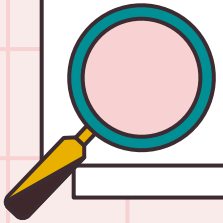
if (current == NULL)
{
    cout << "Vi tri chen khong hop le!" << endl;
    return;
}

Node* newNode = new Node;
newNode->data = x;
newNode->next = current->next;
current->next = newNode;
if (newNode->next == NULL)
{
    a.Tail = newNode;
}
```



3. Tìm một phần tử có Info bằng x

```
// Tìm kiếm một giá trị trong danh sách và trả về vị trí của nó
int Search(Node* head, int x)
{
    Node* current = head;
    int position = 1;
    while (current != NULL && current->data != x)
    {
        current = current->next;
        position++;
    }
    if (current != NULL)
    {
        return position;
    }
    else
    {
        return -1;
    }
}
```





THANKS

Do you have any questions?

