ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

SLIDE BÀI GIẢNG
MÔN

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN, KHU PHỐ 6, PHƯỜNG LINH TRUNG, QUẬN THỦ ĐỨC, TP. HỒ CHÍ MINH
[T] 08 3725 2002 101 | [F] 08 3725 2148 | [W] www.uit.edu.vn | [E] info@uit.edu.vn

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

CHƯƠNG III

CẤU TRÚC DỮ LIỆU ĐỘNG

Nguyễn Trọng Chỉnh
chinhnt@uit.edu.vn
www.uit.edu.vn

MỤC TIÊU CHƯƠNG III

- Hiểu các khái niệm về quản lý bộ nhớ trên C++
- Biết các cấu trúc danh sách liên kết
- Hiểu các thao tác trên danh sách liên kết đơn, liên kết kép và vận dụng vào các danh sách liên kết khác
- Áp dụng danh sách liên kết để giải quyết bài toán trong chương trình C++.

CẤU TRÚC DỮ LIỆU ĐỘNG

- ĐẶT VẤN ĐỀ
- KIỂU DỮ LIỆU CON TRỎ
- DANH SÁCH LIÊN KẾT
- DANH SÁCH ĐƠN
- MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

ĐẶT VẤN ĐỀ

QUẢN LÝ VÙNG NHỚ

Vùng nhớ chương trình được quản lý theo hai cách:

- **Quản lý tự động**:
  - C/C++ tạo vùng nhớ khi khai báo biến, hằng. Các biến, hằng này được gọi chung là biến cấp phát tĩnh.
  - C/C++ tự động giải phóng vùng nhớ khi không còn sử dụng.

ĐẶT VẤN ĐỀ

❖QUẢN LÝ VÙNG NHỚ

- **<u>Quản lý do người lập trình</u>**:
  - Được thực hiện khi gọi các hàm cấp phát vùng nhớ.
  - Vùng nhớ được cấp phát được gọi là biến cấp phát động (biến động)
  - Phải được giải phóng khi hết sử dụng bằng cách gọi các hàm giải phóng vùng nhớ

ĐẶT VẤN ĐỀ

❖BIẾN CẤP PHÁT TĨNH

Là các biến được tạo bằng khai báo. Ví dụ:

```
struct PS{
    int ts, ms;
};
int x, y;
char s[50];
PS p;
```

ĐẶT VẤN ĐỀ

BIẾN CẤP PHÁT TĨNH

Biến cấp phát tĩnh (biến tĩnh, biến nửa tĩnh) có các đặc điểm sau:

- Được khai báo tường minh, có tên gọi.
- Tồn tại trong phạm vi khai báo.
- Được cấp phát trong stack segment hoặc data segment.
- Kích thước không đổi.
  - Ví dụ: kích thước mảng.

ĐẶT VẤN ĐỀ

❖BIẾN CẤP PHÁT ĐỘNG

Là các biến được tạo khi cấp phát vùng nhớ.

Ví dụ:

```
struct PS{
    int ts, ms;
};
```

- Tạo vùng nhớ cho biến kiểu nguyên:

new int;

- Tạo vùng nhớ cho biến kiểu PS:

new PS;

ĐẶT VẤN ĐỀ

BIẾN CẤP PHÁT ĐỘNG

Biến cấp phát động (biến động) có các đặc điểm sau:

- Không được khai báo tường minh, không có tên gọi.
- Cấp phát khi cần sử dụng, khi hết sử dụng phải giải phóng.
- Được cấp phát trong heap segment.
- Kích thước vùng nhớ được cấp phát tùy theo yêu cầu và giới hạn bộ nhớ.

ĐẶT VẤN ĐỀ

❖ NHƯỢC ĐIỂM CỦA CẤP PHÁT TĨNH

- Lãng phí bộ nhớ.

Ví dụ: danh sách nhân viên của một công ty có thể thay đổi.

Giải pháp nếu dùng cấp phát tĩnh:

- Xác định kích thước tối đa n của danh sách
- khai báo mảng với kích thước n

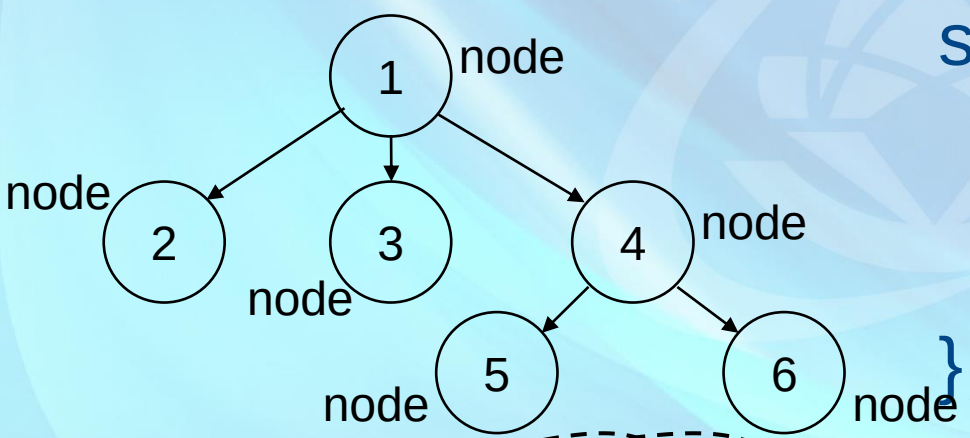
⇒ lãng phí bộ nhớ.

ĐẶT VẤN ĐỀ

- NHƯỢC ĐIỂM CỦA CẤP PHÁT TĨNH

  - Không thể định nghĩa kiểu có cấu trúc đệ quy vì không xác định được kích thước của nó.

  Ví dụ: tổ chức dữ liệu cho cây:




```
struct Node {
    int Key;
    Node left, mid, right;
};
```

KIỂU DỮ LIỆU CON TRỎ

KHÁI NIỆM

Cho kiểu dữ liệu T=<V,O>.

Kiểu con trỏ Tp=<Vp,Op> trỏ đến các biến kiểu T.

⇔ Biến kiểu Tp có giá trị là địa chỉ của các biến kiểu T.

KIỂU DỮ LIỆU CON TRỎ

KHÁI NIỆM

- Vp là miền giá trị của kiểu con trỏ Tp gồm
  - Giá trị NULL(bằng 0)
  - Các địa chỉ của các biến kiểu T.
- Op là các phép toán trên kiểu con trỏ Tp gồm:
  - Tăng địa chỉ (+)
  - Giảm địa chỉ (-)
  - Phân giải địa chỉ (*)
  - Gán giá trị địa chỉ (=)

KIỂU DỮ LIỆU CON TRỎ

❖SỬ DỤNG

- Khai báo kiểu con trỏ:

typedef kiểu_cơ_sở * kiểu_con_trỏ;

- Khai báo biến con trỏ:

kiểu_con_trỏ tên_biến;

hoặc

kiểu_cơ_sở * tên_biến;

Ví dụ:

```
typedef int *IntPtr;
IntPtr a; // tương đương với int *a
```

KIỂU DỮ LIỆU CON TRỎ

❖SỬ DỤNG

- Con trỏ được sử dụng để lưu địa chỉ của biến cấp phát động ⇒ truy xuất biến cấp phát động bằng con trỏ:

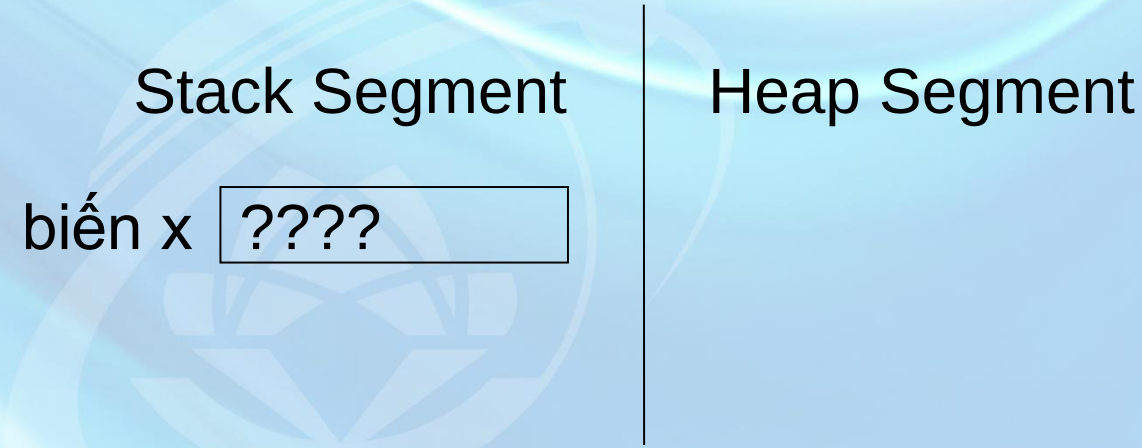
Ví dụ:

```
typedef int *IntPtr;
//....
IntPtr x;
x = new int;
*x = 100;
```

KIỂU DỮ LIỆU CON TRỎ

❖SỬ DỤNG

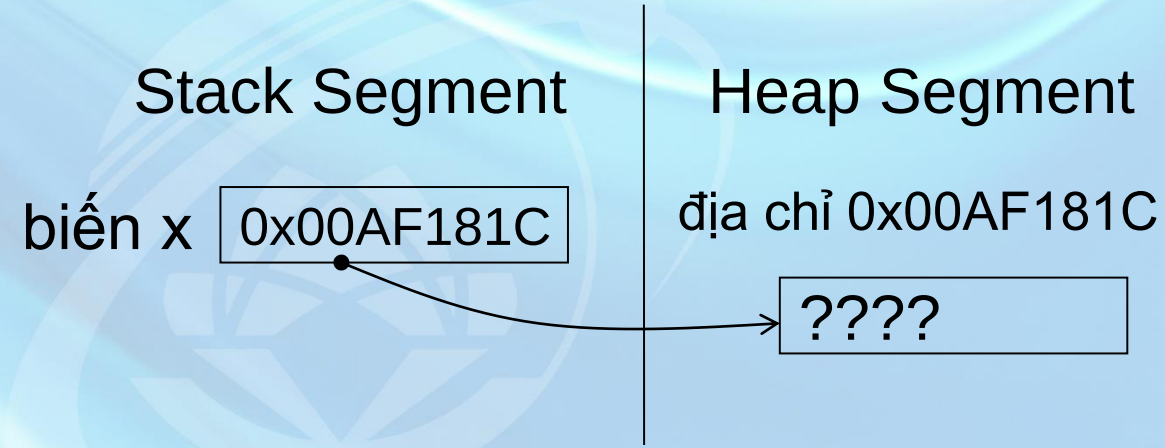
```
typedef int *IntPtr;
//....
IntPtr x;
```



KIỂU DỮ LIỆU CON TRỎ

❖SỬ DỤNG

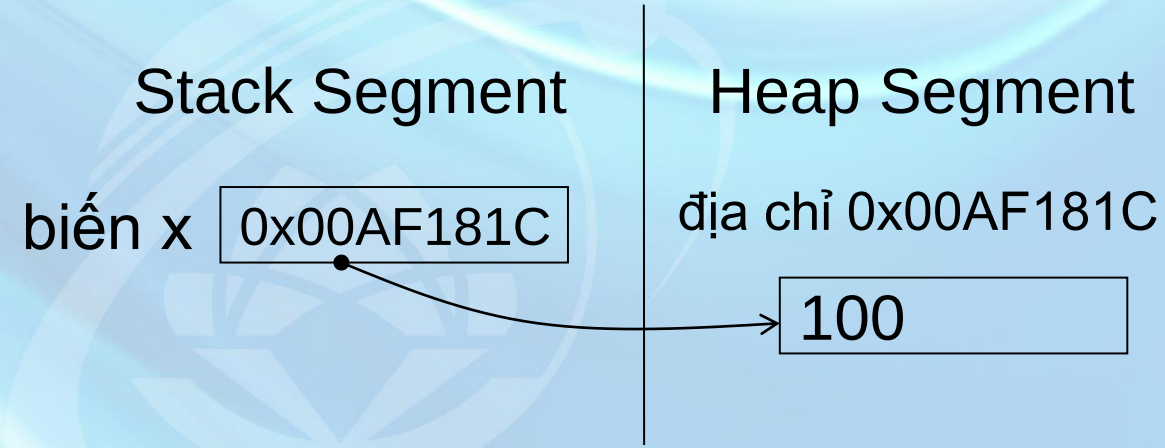
```
typedef int *IntPtr;
//....
IntPtr x;
x = new int;
```



KIỂU DỮ LIỆU CON TRỎ

❖SỬ DỤNG

```
typedef int *IntPtr;
//....
IntPtr x;
x = new int;
*x = 100;
```



Lưu ý: biến con trỏ là một biến cấp phát tĩnh.

KIỂU DỮ LIỆU CON TRỎ

❖CÁC THAO TÁC TRÊN KIỂU CON TRỎ

- Tạo biến cấp phát động để con trỏ quản lý.
  - Cấp phát vùng nhớ cho một biến:

    tên_biến = **new** kiểu;
  - Cấp phát vùng nhớ cho mảng n phần tử:

    tên_biến = **new** kiểu[n];

Kết quả cấp phát là:

- Địa chỉ ô nhớ đầu tiên của vùng nhớ được cấp.
- NULL nếu không cấp phát được.

KIỂU DỮ LIỆU CON TRỎ

CÁC THAO TÁC TRÊN KIỂU CON TRỎ

- Giải phóng biến cấp phát động do con trỏ quản lý.
  - Nếu p là một đối tượng đơn lẻ

    delete p;
  - Nếu p là một mảng các đối tượng

    delete [] p;

KIỂU DỮ LIỆU CON TRỎ

❖CÁC THAO TÁC TRÊN KIỂU CON TRỎ

- Tăng địa chỉ vùng nhớ do con trỏ quản lý lên n lần kích thước của kiểu con trỏ:

Ví dụ:

```
int *p = new int;  // giả sử giá trị của p là 1
p = p + 4;         // giá trị của p là 1+4*4=17
```

Lưu ý: *(p + i) tương đương với p[i]

KIỂU DỮ LIỆU CON TRỎ

❖CÁC THAO TÁC TRÊN KIỂU CON TRỎ

- Giảm địa chỉ vùng nhớ do con trỏ quản lý xuống n lần kích thước của kiểu con trỏ:

Ví dụ:

```
int *p = new int;  // giả sử giá trị của p là 17
p--;               // giá trị của p là 17-1*4=13.
```

KIỂU DỮ LIỆU CON TRỎ

❖CÁC THAO TÁC TRÊN KIỂU CON TRỎ

- Phân giải địa chỉ con trỏ (dereference):
  - Dùng phép toán *
  - Truy cập đến biến cấp phát động.

Ví dụ:

```
int *p;
p = new int;    // tạo một biến cấp phát động cho p
*p = 100;       // gán 100 cho vùng được cấp phát
*p *= 2;        // nhân 2 vùng được cấp phát
cout << *p;     // in ra giá trị vùng được cấp phát
```

KIỂU DỮ LIỆU CON TRỎ

❖CÁC THAO TÁC TRÊN KIỂU CON TRỎ

Ví dụ:

Viết chương trình nhập vào dãy số gồm n giá trị nguyên (n < 1000 và được nhập từ bàn phím), in ra màn hình các giá trị đã nhập theo thứ tự tăng dần.

KIỂU DỮ LIỆU CON TRỎ

```
#include <iostream>
#include <cstdlib>
using namespace std;
typedef int *DaySo;
void BubbleSort(DaySo A, int n) {
    int i, j, x;
    for (i = 0; i < n - 1; i++)
        for (j = n - 1; j > i; j--)
            if (A[j] < A[j-1])
                {x = A[j]; A[j] = A[j - 1]; A[j-1] = x;}
}
```

KIỂU DỮ LIỆU CON TRỎ

```
int main(char **arg, int c) {
    int n, i;
    DaySo A;
    cin >> n;
    A = new int[n];
    if (A == NULL) return EXIT_FAILURE;
    for (i = 0; i < n; i++)
        cin >> A[i];
    cout << endl;
    BubbleSort(A, n);
```

KIỂU DỮ LIỆU CON TRỎ

```
    for (i = 0; i < n; i++)
        cout << A[i] << ' ';
    cout << endl;
    delete [] A;
    return EXIT_SUCCESS;
}
```

DANH SÁCH LIÊN KẾT

KHÁI NIỆM DANH SÁCH

Danh sách là một tập hợp các phần tử:

- Cùng kiểu dữ liệu
- Có duy nhất một phần tử liền trước nó
- Có duy nhất một phần tử liền sau nó.

Ví dụ: danh sách thí sinh

STT	Họ Tên	....
1	Nguyen Van A	....
2	Tran Thi B	....
3	Vo Van C	....

DANH SÁCH LIÊN KẾT

❖PHÂN LOẠI DANH SÁCH

- **Danh sách đặc (condensed list)**

Là danh sách được tổ chức sao cho hai phần tử liên tiếp nhau có vị trí vùng nhớ liên tiếp nhau trên bộ nhớ.

Ví dụ: Mảng cấp phát tĩnh, mảng cấp phát động.

DANH SÁCH LIÊN KẾT

❖PHÂN LOẠI DANH SÁCH

- **Danh sách đặc (condensed list)**

<u>Đặc điểm</u>:

- Phần tử kế tiếp A_{i+1} của phần tử A_i được xác định ngầm nhờ giá trị địa chỉ như sau:

  Địa chỉ A_{i+1} = Địa chỉ A_i + kích thước kiểu của A_i;

DANH SÁCH LIÊN KẾT

❖PHÂN LOẠI DANH SÁCH

- **Danh sách đặc (condensed list)**

Ưu điểm:

- Truy xuất trực tiếp, ngẫu nhiên (random access)
- Nhanh.

Ví dụ:

int a[10];

a[1] = 0; a[9] = 2;

DANH SÁCH LIÊN KẾT

❖PHÂN LOẠI DANH SÁCH

- **Danh sách đặc (condensed list)**

<u>Nhược điểm điểm</u>:

- Các thao tác thêm và xóa phần tử không hiệu quả.
- Sử dụng bộ nhớ không hiệu quả.
- Số lượng phần tử thường cố định.

DANH SÁCH LIÊN KẾT

❖PHÂN LOẠI DANH SÁCH

- **Danh sách liên kết (linked list)**

Là danh sách mà mỗi phần tử cần phải lưu trữ thông tin của nó và địa chỉ của phần tử liền kề với nó.

DANH SÁCH LIÊN KẾT

❖PHÂN LOẠI DANH SÁCH

- **Danh sách liên kết (linked list)**

Đặc điểm:

- Phần tử kế tiếp A_{i+1} của phần tử A_i được xác định tường minh bằng địa chỉ của nó được lưu trong A_i

DANH SÁCH LIÊN KẾT

❖PHÂN LOẠI DANH SÁCH

- **Danh sách liên kết (linked list)**

<u>Ưu điểm</u>:

- Thao tác thêm, xóa được thực hiện dễ dàng.
- Sử dụng hiệu quả bộ nhớ
- Số lượng phần tử thay đổi dễ dàng.

DANH SÁCH LIÊN KẾT

❖PHÂN LOẠI DANH SÁCH

- **Danh sách liên kết (linked list)**

Nhược điểm:

- Truy xuất tuần tự (serial access)
- Chậm hơn danh sách đặc.

DANH SÁCH LIÊN KẾT

❖CÁC LOẠI DANH SÁCH LIÊN KẾT

- **Danh sách liên kết đơn (singly linked linear list)**

Mỗi phần tử cần

- Lưu trữ thông tin của phần tử
- Lưu địa chỉ của phần tử liền sau

DANH SÁCH LIÊN KẾT

❖CÁC LOẠI DANH SÁCH LIÊN KẾT

- **Danh sách liên kết kép (doubly linked linear list)**

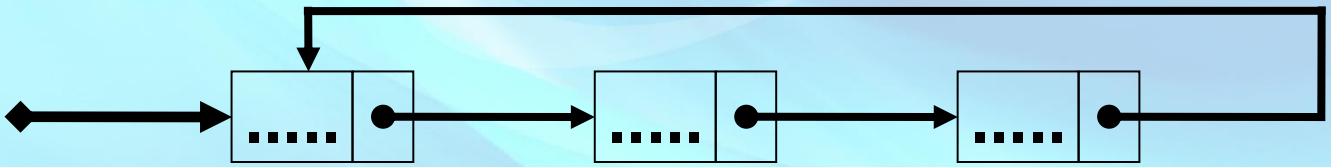
Mỗi phần tử:

- Lưu thông tin của nó
- Địa chỉ của hai phần từ liền trước và liền sau nó.

DANH SÁCH LIÊN KẾT

❖CÁC LOẠI DANH SÁCH LIÊN KẾT

- **Danh sách liên kết đơn vòng (circular singly linked list)**
  - Là danh sách liên kết đơn
  - Phần tử cuối danh sách liên kết với phần tử đầu danh sách



DANH SÁCH LIÊN KẾT

❖CÁC LOẠI DANH SÁCH LIÊN KẾT

- **Danh sách liên kết kép vòng (circular doubly linked list)**
  - Là danh sách liên kết kép
  - Phần tử cuối danh sách liên kết với phần tử đầu danh sách.