Lập trình

Chương 3: Cấu trúc dữ liệu

2/21/2019

Nội dung bài giảng

3.1 Giới thiệu chung

3.2 Mảng và quản lý bộ nhớ động

3.4 Nội dung và mục đích của cấu trúc dữ liệu

3.4 Cài đặt một số cấu trúc với C++

3.1 Giới thiệu chung

- *Phần lớn các bài toán trong thực tế liên quan tới các dữ liệu phức hợp, những kiểu dữ liệu cơ bản trong ngôn ngữ lập trình không đủ biểu diễn*
- *Ví dụ:*
  - *Dữ liệu sinh viên: Họ tên, ngày sinh, quê quán, mã số SV,...*
  - *Mô hình hàm truyền: Đa thức tử số, đa thức mẫu số*
  - *Mô hình trạng thái: Các ma trận A, B, C, D*
  - *Đối tượng đồ họa: Kích thước, màu sắc, đường nét, phông chữ, ...*
- *Phương pháp biểu diễn dữ liệu: định nghĩa kiểu dữ liệu mới sử dụng cấu trúc (struct, class, union, ...)*

Vấn đề: Biểu diễn tập hợp dữ liệu

- *Đa số những dữ liệu thuộc một ứng dụng có liên quan với nhau => cần biểu diễn trong một tập hợp có cấu trúc, ví dụ:*
  - *Danh sách sinh viên: Các dữ liệu sinh viên được sắp xếp theo thứ tự Alphabet*
  - *Đối tượng đồ họa: Một cửa sổ bao gồm nhiều đối tượng đồ họa, một bản vẽ cũng bao gồm nhiều đối tượng đồ họa*
- *Thông thường, các dữ liệu trong một tập hợp có cùng kiểu, hoặc ít ra là tương thích kiểu với nhau*
- *Kiểu mảng không phải bao giờ cũng phù hợp!*

Vấn đề: Quản lý dữ liệu

- *Sử dụng kết hợp một cách khéo léo kiểu cấu trúc và kiểu mảng đủ để biểu diễn các tập hợp dữ liệu bất kỳ*
- *Các giải thuật (hàm) thao tác với dữ liệu, nhằm quản lý dữ liệu một cách hiệu quả:*
  - *Bổ sung một mục dữ liệu mới vào một danh sách, một bảng, một tập hợp, ...*
  - *Xóa một mục dữ liệu trong một danh sách, bảng, tập hợp,..*
  - *Tìm một mục dữ liệu trong một danh sách, bảng tập hợp,... theo một tiêu chuẩn cụ thể*
  - *Sắp xếp một danh sách theo một tiêu chuẩn nào đó*
  - *....*

Quản lý DL thế nào là hiệu quả?

- *Tiết kiệm bộ nhớ*
- *Truy nhập nhanh, thuận tiện: Thời gian cần cho bổ sung, tìm kiếm và xóa bỏ các mục dữ liệu phải ngắn*
- *Linh hoạt: Số lượng các mục dữ liệu không (hoặc ít) bị hạn chế cố định, không cần biết trước khi tạo cấu trúc, phù hợp với cả bài toán nhỏ và lớn*
- *Hiệu quả quản lý dữ liệu phụ thuộc vào*
  - *Cấu trúc dữ liệu được sử dụng*
  - *Giải thuật được áp dụng cho bổ sung, tìm kiếm, sắp xếp, xóa bỏ*

Các cấu trúc dữ liệu thông dụng

- *Mảng (nghĩa rộng): Tập hợp các dữ liệu có thể truy nhập tùy ý theo chỉ số*
- *Danh sách (list): Tập hợp các dữ liệu được móc nối đôi một với nhau và có thể truy nhập tuần tự*
- *Cây (tree): Tập hợp các dữ liệu được móc nối với nhau theo cấu trúc cây, có thể truy nhập tuần tự từ gốc*
- *Hàng đợi (queue): Tập hợp các dữ liệu có sắp xếp tuần tự, chỉ bổ sung vào từ một đầu và lấy ra từ đầu còn lại*
- *Ngăn xếp (stack): Tập hợp các dữ liệu được sắp xếp tuần tự, chỉ truy nhập được từ một đầu*
- *Bộ nhớ vòng (ring buffer): Tương tự như hàng đợi, nhưng dung lượng có hạn, nếu hết chỗ sẽ được ghi quay vòng*

3.2 Mảng và quản lý bộ nhớ động

- *Mảng cho phép biểu diễn và quản lý dữ liệu một cách khá hiệu quả:*
  - *Đọc và ghi dữ liệu rất nhanh qua chỉ số hoặc qua địa chỉ*
  - *Tiết kiệm bộ nhớ*
- *Các vấn đề của mảng tĩnh:*

  `VD: Student student_list[100];`
  - *Số phần tử phải là hằng số (biết trước khi biên dịch, người sử dụng không thể nhập số phần tử, không thể cho số phần từ là một biến) => kém linh hoạt*
  - *Chiếm chỗ cứng trong ngăn xếp (đối với biến cục bộ) hoặc trong bộ nhớ dữ liệu chương trình (đối với biến toàn cục) => sử dụng bộ nhớ kém hiệu quả, kém linh hoạt*

Mảng động

- *Mảng động là một mảng được cấp phát bộ nhớ theo yêu cầu, trong khi chương trình chạy*

```
#include <stdlib.h>     /* C */
int n = 50;
...
float*  p1= (float*) malloc(n*sizeof(float)); /* C */
double* p2= new double[n];    // C++
```

- *Sử dụng con trỏ để quản lý mảng động: Cách sử dụng không khác so với mảng tĩnh*

```
p1[0] = 1.0;
p2[0] = 2.0;
```

- *Sau khi sử dụng xong => giải phóng bộ nhớ:*

```
free(p1);          /* C */
delete [] p2;      // C++
```

Cấp phát và giải phóng bô nhớ đông

- *C:*
  - *Hàm* `malloc()` *yêu cầu tham số là* ***số byte****, trả về con trỏ không kiểu (*`void*`*) mang địa chỉ vùng nhớ mới được cấp phát (nằm trong heap), trả về 0 nếu không thành công.*
  - *Hàm* `free()` *yêu cầu tham số là con trỏ không kiểu (*`void*`*), giải phóng vùng nhớ có địa chỉ đưa vào*
- *C++:*
  - *Toán tử* `new` *chấp nhận kiểu dữ liệu phần tử kèm theo số lượng phần tử của mảng cần cấp phát bộ nhớ (trong vùng heap), trả về con trỏ có kiểu, trả về 0 nếu không thành công.*
  - *Toán tử* `delete[]` *yêu cầu tham số là con trỏ có kiểu.*
  - *Toán tử* `new` *và* `delete` *còn có thể áp dụng cho cấp phát và giải phóng bộ nhớ cho một biến đơn, một đối tượng chứ không nhất thiết phải một mảng.*

Một số điều cần lưu ý

- *Con trỏ có vai trò quản lý mảng (động), chứ con trỏ không phải là mảng (động)*
- *Cấp phát bộ nhớ và giải phóng bộ nhớ chứ không phải cấp phát con trỏ và giải phóng con trỏ*
- *Chỉ giải phóng bộ nhớ một lần*
- *Ví dụ:*

```
int* p;
p[0] = 1;                  // ??
new(p);                    // ??
p = new int[100];          // OK
p[0] = 1;                  // OK
int* p2=p;                 // OK
delete[] p2;               // OK
p[0] = 1;                  //??
delete[] p;                //??
p = new int[50];           // OK, new array
...
```

Cấp phát bộ nhớ động cho biến đơn

- ***Ý nghĩa**: Các đối tượng có thể được tạo ra động, trong khi chương trình chạy (bổ sung sinh viên vào danh sách, vẽ thêm một hình trong bản vẽ, bổ sung một khâu trong hệ thống,...)*
- *Cú pháp*

```
int* p = new int;
*p = 1;
p[0]= 2;                    // giong nhu tren
p[1]= 1;                    // ??
int* p2 = new int(1);       // có khởi tạo
delete p;
delete p2;
Student* ps = new Student;
ps->code = 1000;
...
delete ps;
```

Ý nghĩa của sử dung bô nhớ đông

- *Hiệu suất:*
  - *Bộ nhớ được cấp phát đủ dung lượng theo yêu cầu và khi được yêu cầu trong khi chương trình đã chạy*
  - *Bộ nhớ được cấp phát nằm trong vùng nhớ tự do còn lại của máy tính (heap), chỉ phụ thuộc vào dung lượng bộ nhớ của máy tính*
  - *Bộ nhớ có thể được giải phóng khi không sử dụng tiếp.*
- *Linh hoạt:*
  - *Thời gian "sống" của bộ nhớ được cấp phát động có thể kéo dài hơn thời gian "sống" của thực thể cấp phát nó.*
  - *Có thể một hàm gọi lệnh cấp phát bộ nhớ, nhưng một hàm khác giải phóng bộ nhớ.*
  - *Sự linh hoạt cũng dễ dẫn đến những lỗi "rò rỉ bộ nhớ".*

Ví dụ sử dụng bộ nhớ động trong hàm

```
Date* createDateList(int n) {
        Date* p = new Date[n];
        return p;
}
void main() {
        int n;
        cout << "Enter the number of your national
holidays:";
        cin >> n;
        Date* date_list = createDateList(n);
        for (int i=0; i < n; ++i) {
                ...
        }
        for (....) { cout << ....}
        delete [] date_list;
}
```

Các thuật toán trên mảng

- *Các thuật toán sắp xếp*
- *Thuật toán tìm kiếm chia đôi (tìm kiếm nhị phân)*

Các thuật toán sắp xếp trên mảng

- *Các giải thuật sắp xếp cơ bản*
  - *Sắp xếp chọn (selection-sort)*
  - *Sắp xếp nổi bọt (bubble, exchange-sort)*
  - *Sắp xếp chèn (insertion-sort)*
- *Các giải thuật sắp xếp nâng cao-Sắp xếp nhanh*
  - *Sắp xếp nhanh (quick-sort)*
  - *Sắp xếp vun đống (heap-sort)*
  - *Sắp xếp hòa trộn (merge-sort)*
- *Quy ước:*
  - *Giả sử các phần tử khóa cần sắp xếp là các số*
  - *Sắp xếp thực hiện theo thứ tự từ nhỏ đến lớn*

Sắp xếp chọn (selection-sort)

- *Nguyên tắc*
  - *Lượt 1: tìm khóa nhỏ nhất trong n khóa đưa lên vị trí thứ nhất (đổi chỗ với khóa đầu tiên)*
  - *Lượt 2: tìm khóa nhỏ nhất trong n-1 khoá còn lại đưa lên vị trí thứ 2 (đổi chỗ với khóa thứ 2)*
  - *...*
  - *Lượt i: tìm khóa nhỏ nhất trong (n-i+1) khoá còn lại đưa lên vị trí thứ i (đổi chỗ với khóa thứ i)*
- *Ví dụ:*
  - *Sắp xếp:*

    25, 36, 31, 49, 16, 70, 88, 60
    16, *36, 31, 49,* ***25***, *70, 88, 60*
    16, 25, *31, 49,* ***36***, *70, 88, 60*
    16, 25, 31, *49, 36, 70, 88, 60*
    ...

Sắp xếp chèn (Insertion-sort)

- *Nguyên tắc*
  - *Giả sử đoạn đầu đã được sắp xếp, thêm một khóa mới => tìm vị trí thích hợp cho khóa mới. Cứ làm như vậy từ 2 đến n khóa.*
  - *So sánh và sắp xếp dần từ đằng sau danh sách khóa.*
- *Phân biệt hai trường hợp*
  - *Trường hợp cần đưa lần lượt các khóa vào: nhập các khóa lần lượt vào các ô nhớ. Dãy khóa: 25, 36, 31, 49, 16, 70, 88, 60*

    1: 25

    2: 25, 36

    3: 25, 31, 36

    4: 25, 31, 36, 49

    5: 16, 25, 31, 36, 49

    6: ...
  - *Trường hợp các khóa đã có và cần sắp xếp lại: dùng ô nhớ phụ (biến) X để lưu tạm các giá trị cần dịch chuyển. Hướng dịch chuyển: dần về đầu.*
  - *1: **25**, 36, 31, 49, 16, 70, 88, 60*
  - *2: **25, 36**, 31, 49, 16, 70, 88, 60*
  - *3: **25, 31, 36**, 49, 16, 70, 88, 60*
  - *4: **25, 31, 36, 49**, 16, 70, 88, 60*
  - *5: **16, 25, 31, 36, 49**, 70, 88, 60*
  - *6: ...*

Sắp xếp nổi bọt (Bubble-sort)

- *Nguyên tắc*
  - *Duyệt bảng khoá (danh sách khoá) từ đáy lên đỉnh*
  - *Dọc đường nếu thứ tự 2 khoá liền kế không đúng => đổi chỗ*
- *Ví dụ:*
  - *1: 25, 36, 31, 60, 16, 88, 49, 70*
  - *2: **16**, 25, 36, 31, 60, **49**, 88, 70*
  - *3: 16, 25, **31, 36**, **49**, **60**, **70**, **88***
- *Nhận xét*
  - *Khoá nhỏ sẽ nổi dần lên sau mỗi lần duyệt => **"nổi bọt"***
  - *Sau một vài lần (không cần chạy n bước), danh sách khoá đã có thể được sắp xếp => Có thể cải tiến thuật toán, dùng 1 biến lưu trạng thái, nếu không còn gì thay đổi (không cần đổi chỗ) => ngừng*

Sắp xếp nhanh (quick-sort) hay Sắp xếp phân đoạn

- *Nguyên tắc*
  - *Chiến lược chia để trị*
  - *Chọn mốc (bất kỳ) để phân thành từng đoạn (partition) (2 đoạn)*
  - *Khoá của mốc sẽ đứng giữa: Left Keys <= key(mốc) <= Right Keys*
- *Thuật toán*
  - *1: Quy ước: luôn chọn phần tử đầu tiên làm mốc.*
  - *2: Tìm vị trí thực của khóa mốc để phân thành 2 đoạn:*
    - *1: Dùng 2 chỉ số: i, j chạy từ hai đầu DS. Chạy i từ đầu DS trong khi khóa còn nhỏ hơn khóa mốc*
    - *2: Nếu khóa >= khóa mốc: Lưu phần tử hiện tại = X và chạy j.*
    - *3: Chạy j trong khi khóa lớn hơn khóa mốc*
    - *4: Nếu khóa <= khóa mốc: dừng và đổi chỗ phần tử hiện tại cho X*
    - *5: Tiếp tục thực hiện như vậy cho đến khi i>=j thì đổi chỗ Kj cho khóa mốc. Lúc đó khóa mốc sẽ nằm đúng vị trí.*
  - *3: Làm giống như vậy cho các đoạn tiếp theo*

Sắp xếp nhanh (...)

- *Ví dụ:*
  - *0:* ***25****, 36, 31, 49, 16, 70, 88, 60*
  - *1.1:* ***25****, 36, 31, 49, 16, 70, 88, 60*
  - *1.2:* ***25****,* ***16****, 31, 49,* ***36****, 70, 88, 60*
  - *1.3:* ***(16), 25, (31, 49, 36, 70, 88, 60)***
  - *2.1:* ***(16)***
  - *3.1:* ***31****, 49, 36, 70, 88, 60*
  - *3.2: ....*

Sắp xếp nhanh (...)

- *Nhận xét*
  - *Vấn đề chọn mốc:*
    - *Nếu luôn chọn mốc có giá trị khóa = min => một lần chỉ chuyển được 1 phần tử: giống bubble-sorting*
    - *Nếu mốc có giá trị khóa trung bình => chia được làm 2 đoạn có độ dài tương đương.*
  - *Vấn đề phối hợp với các giải thuật đơn giản khác:*
    - *Giải thuật tương đối phức tạp => khi các đoạn cần sắp xếp đủ nhỏ, ta phối hợp với các giải thuật sắp xếp đơn giản đã nêu trên.*

Sắp xếp vun đống (heap-sort)

- *Nguyên tắc*
  - *Dùng cấu trúc **cây NP hoàn chỉnh** lưu trữ các khóa bằng CTLT tuần tự*

  (Cây NP hoàn chỉnh là cây nhị phân mà tất cả các nút ở chiều cao h-1 đều có 2 con, các nút có chiều cao h thì dồn về bên trái)
  - *Khái niệm **"đống"** (heap):*
    - *là cây nhị phân hoàn chỉnh*
    - *luôn có: key(parent) >= key(child)*
  - *Các giai đoạn:*
    - *Biến đổi cây NP thành **"đống"** (heap)*
      - *Lá = heap*
      - *Vun lá thành đống (bottom-up)*
    - *Sắp xếp các khóa:*
      - *Tìm khóa lớn nhất, đưa ra ngoài*
      - *Vun lại thành **"đống"** (heap) đối với cây còn lại*

Sắp xếp vun đống (...)

- *Ví dụ*
  - *0: 25, 36, 31, 49, 16, 70, 88, 60*
  - *1: Tạo cây NP*
  - *2: Vun đống từ lá*
  - *3: Lấy ra số max, tráo với phần tử hiện tại*
  - *4: Sắp lại cây thành heap.*
  - *5: Lặp lại bước 3, 4 cho đến hết.*

Sắp xếp trộn (merge-sort)

- *Nguyên tắc*
  - *1 phần tử (lá) coi như là đã được sắp xếp*
  - *L1 được sắp xếp*
  - *L2 được sắp xếp*
  - *Trộn L1, L2 => L*
- *Ví dụ*
  - *25, 36, 31, 49, 16, 70, 88, 60*
  - *1: [25], [36], [31], [49], [16], [70], [88], [60]*
  - *2: [25, 36], [31, 49], [16, 70], [60, 88]*
  - *3: [25, 31, 36, 49], [16, 60, 70, 88]*
  - *4: [16, 31, 36, 49, 60, 70, 88]*

Nhận xét các giải thuật sắp xếp

- *Các giải thuật sắp xếp đơn giản*
  - *Sắp xếp chọn (selection-sort):*
    - $T_{min} = T_{max} = T_{tb} = T(n) = O(n^2)$
  - *Sắp xếp chèn (insertion-sort)*
    - $T_{min} = O(n)$; $T_{max} = O(n^2)$; $T_{tb} = O(n^2)$
  - *Sắp xếp nổi bọt (bubble, exchange-sort)*
    - $T_{min} = T_{max} = T_{tb} = T(n) = O(n^2)$
- *Các giải thuật sắp xếp nâng cao*
  - *Sắp xếp nhanh (quick-sort)*
    - $T_{max} = O(n^2)$; $T_{tb} = O(n \log n)$
  - *Sắp xếp vun đống (heap-sort)*
    - $T_{max} = T_{tb} = O(n \log n)$
  - *Sắp xếp hòa trộn (merge-sort)*
    - $T_{max} = T_{tb} = O(n \log n)$

Giải thuật tìm kiếm chia đôi

- *Ý tưởng giải thuật*
  - *Giả sử các khoá được sắp (VD: theo thứ tự nhỏ đến lớn) thành 1 đoạn*
  - *So sách giá trị cần tìm với khóa ở giữa*
    - *Nếu nhỏ hơn: tìm kiếm NP với đoạn bên trái*
    - *Nếu bằng: dừng lại*
    - *Nếu lớn hơn: tìm kiếm NP với đoạn bên phải*

3. 3 Mục đích và nội dung của CTDL

- *Mục đích:*
  - *Môn học CTDL & giải thuật dành cho các sinh viên đã có những kiến thức cơ bản về lập trình và thành thạo ít nhất một trong số các ngôn ngữ lập trình cơ bản như Pascal, C, C++,..*
  - *Củng cố và nâng cao kiến thức cơ bản về cấu trúc dữ liệu và giải thuật của ngành khoa học máy tính.*
  - *Tăng cường khả năng phân tích, thiết kế và cài đặt các chương trình cho máy tính.*
  - *Nâng cao khả năng tư duy trừu tượng và sự khái quát khi giải quyết các bài toán thực tế bằng máy tính*

Mục đích và nội dung của CTDL

- *Nội dung:*
  - *Trình bầy các phương pháp phân tích và thiết kế một chương trình.*
  - *Giới thiệu các cấu trúc dữ liệu từ đơn giản (các cấu trúc tuyến tính như : mảng, danh sách) đến phức tạp (các cấu trúc phi tuyến như: cây, đồ thị) và các thao tác cơ bản tương ứng trên các cấu trúc dữ liệu.*
  - *Tìm hiểu các giải thuật từ cơ bản như các giải thuật sắp xếp, tìm kiếm, đến một số giải thuật nâng cao như các giải thuật đệ quy, các giải thuật trên các cấu trúc dữ liệu cây, đồ thị.*

Ví dụ minh họa

- *Yêu cầu: Viết một chương trình quản lý danh sách sinh viên của một lớp. Mỗi sinh viên gồm các thuộc tính: Mã số, Họ tên, Ngày sinh, Địa chỉ, Tên lớp, Môn thi, Điểm thi. Chương trình cần thực hiện các công việc sau:*
  - *Cập nhật thông tin cho từng sinh viên trong danh sách, tức là có thể bổ sung, loại bỏ, hay cập nhật các thuộc tính một sinh viên trong danh sách*
  - *Sắp xếp danh sách theo một trật tự nhất định, như theo Họ tên theo trật tự từ A-Z,v.v*
  - *Tìm kiếm một sinh viên theo một tiêu chuẩn nào đó, ví như tìm theo Họ tên, hay theo Mã số,v.v*
  - *In nội dung của danh sách*
  - *….*

Ví dụ minh họa

- *Phân tích yêu cầu trên: có 2 nhiệm vụ chính mà chúng ta cần làm trước khi xây dựng được chương trình trên:*
  - *Nắm được cách **tổ chức** và **cài đặt** cho cấu trúc danh sách sinh viên nói riêng, khái quát hơn là cho cấu trúc danh sách nói chung → **cần nắm được cấu trúc dữ liệu***
  - *Nắm được **ý tưởng** và **cách cài đặt** cho các thao tác cơ bản như tìm kiếm, sắp xếp → **cần nắm được giải thuật***

Các khái niệm cơ bản về CTDL và giải thuật

- ***Giải thuật*** *(algorithm):*
  - *Là một* ***đặc tả*** *chính xác và không nhập nhằng về một chuỗi các bước có thể được thực hiện một các tự động, để cuối cùng ta có thể thu được các kết quả mong muốn.*
  - *Đặc tả (specification) : bản mô tả chi tiết và đầy đủ về một đối tượng hay một vấn đề*

Giải thuật

- *Một số yêu cầu của giải thuật*
  - *Đúng đắn,*
  - *Rõ ràng*
  - *Phải kết thúc sau một số hữu hạn bước thực hiện,*
  - *Có mô tả các đối tượng dữ liệu mà thuật toán sẽ thao tác như dữ liệu vào (nguồn), dữ liệu ra (đích) và các dữ liệu trung gian,*
  - *Thời gian thực hiện phải hợp lý.*

Dữ liệu

- ***Dữ liệu*** *(data):*
  - *Nó là các đối tượng mà thuật toán sẽ sử dụng để đạt được kết quả mong muốn. Nó cũng được dùng để biểu diễn cho các thông tin của bài boán như: các thông tin vào, thông tin ra (kết quả) và các các thông tin trung gian nếu cần.*

Dữ liệu

- *Dữ liệu gồm có hai mặt:*
  - ***Mặt tĩnh*** *(static): xác định kiểu dữ liệu (data type). Kiểu dữ liệu cho ta biết cách tổ chức dữ liệu cũng như tập các giá trị mà một đối tượng dữ liệu có thể nhận, hay miền giá trị của nó. Ví dụ như kiểu số nguyên, kiểu số thực,..*
  - ***Mặt động*** *(dynamic): là trạng thái của dữ liệu như tồn tại hay không tồn tại, sẵn sàng hay không sẵn sàng. Nếu dữ liệu đang tồn tại thì mặt động của nó còn thể hiện ở giá trị cụ thể của dữ liệu tại từng thời điểm. Trạng thái hay giá trị của dữ liệu sẽ bị thay đổi khi xuất hiện những sự kiện, thao tác tác động lên nó.*

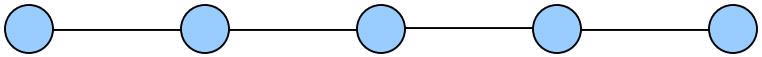
Cấu trúc dữ liệu

- ***Cấu trúc dữ liệu*** *(data structure) :*
  - *Là kiểu dữ liệu mà bên trong nó có chứa nhiều thành phần dữ liệu và các thành phần dữ liệu đấy được tổ chức theo một cấu trúc nào đó. Nó dùng để biểu diễn cho các thông tin có cấu trúc của bài toán. Cấu trúc dữ liệu thể hiện khía cạnh logic của dữ liệu.*
  - *Còn các dữ liệu không có cấu trúc được gọi là các* ***dữ liệu vô hướng*** *hay các* ***dữ liệu đơn giản****. VD: các kiểu dữ liệu số nguyên (integer), số thực (real), logic (boolean) là các kiểu dữ liệu đơn giản.*

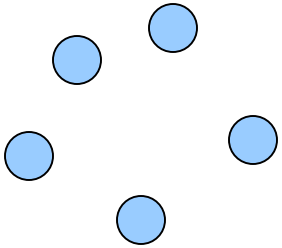
Cấu trúc dữ liệu

- *Có hai loại cấu trúc dữ liệu chính:*
  - ***Cấu trúc tuyến tính**: là cấu trúc dữ liệu mà các phần tử bên trong nó luôn được bố trí theo một trật tự tuyến tính hay trật tự trước sau. Đây là loại cấu trúc dữ liệu đơn giản nhất. Ví dụ :mảng, danh sách.*
  - ***Cấu trúc phi tuyến**: là các CTDL mà các thành phần bên trong không còn được bố trí theo trật tự tuyến tính mà theo các cấu trúc khác. Ví dụ: tập hợp (không có trật tự), cấu trúc cây (cấu trúc phân cấp), đồ thị (cấu trúc đa hướng).*

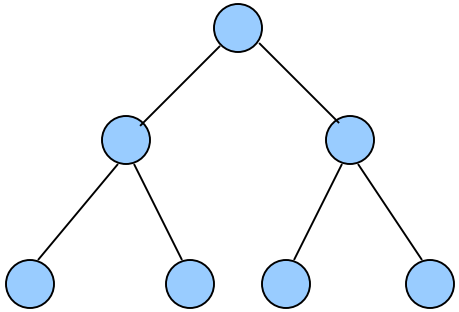
Hình minh họa: các loại CTDL



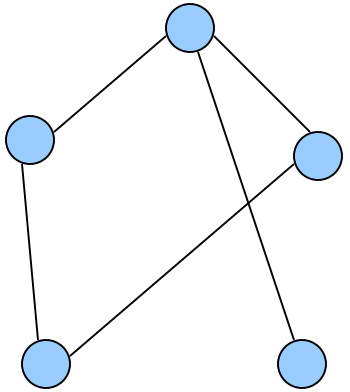
Danh sách



Tập hợp



Cây



Đồ thị

Cấu trúc lưu trữ (storage structure)

- *Cấu trúc lưu trữ của một cấu trúc dữ liệu thể hiện khía cạnh vật lý (cài đặt) của cấu trúc dữ liệu đó.*
- *Về nguyên tắc, nó là một trong số các cách tổ chức lưu trữ của máy tính*
- *Tuy nhiên trong thực tế sử dụng, cấu trúc lưu trữ thường được hiểu là cấu trúc kiểu dữ liệu mà một ngôn ngữ lập trình hỗ trợ, và số lượng các cấu trúc lưu trữ thường là số lượng các kiểu dữ liệu của ngôn ngữ lập trình đó*

Cấu trúc lưu trữ

Có hai loại cấu trúc lưu trữ chính:

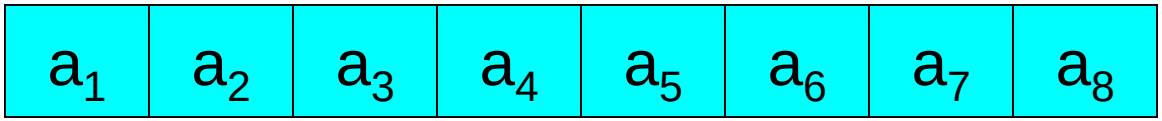
- ***Cấu trúc lưu trữ trong**: là CTLT nằm ở bộ nhớ trong (bộ nhớ chính) của máy tính. CTLT này có đặc điểm là tương đối đơn giản, dễ tổ chức và tốc độ thao tác rất nhanh. Tuy nhiên, CTLT này có nhược điểm là không có tính lưu tồn (persistence), và kích thước khá hạn chế.*
- ***Cấu trúc lưu trữ ngoài**: là CTLT nằm ở bộ nhớ ngoài (bộ nhớ phụ). CTLT ngoài thường có cấu trúc phức tạp và tốc độ thao tác chậm hơn rất nhiều so với CTLT trong, nhưng CTLT này có tính lưu tồn và cho phép chúng ta lưu trữ các dữ liệu có kích thước rất lớn.*

Cấu trúc lưu trữ trong

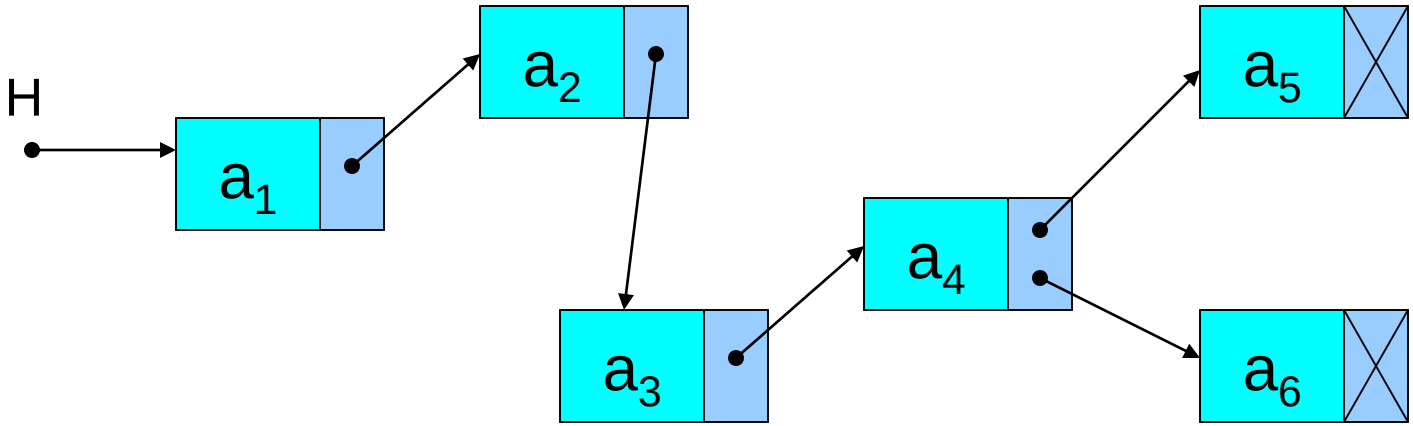
Cấu trúc lưu trữ trong lại được chia làm hai loại:

- ***Cấu trúc lưu trữ tĩnh**: là CTLT mà kích thước dữ liệu luôn cố định. Cấu trúc này còn được gọi là CTLT tuần tự.*
- ***Cấu trúc lưu trữ động**: là CTLT mà kích thước dữ liệu có thể thay đổi trong khi chạy chương trình. Cấu trúc này còn được gọi là cấu trúc con trỏ hay móc nối.*

Hình minh họa: các loại CTLT trong



Cấu trúc tĩnh



Cấu trúc động

Một số đặc điểm của các CTLT trong

- *CTLT tĩnh:*
  - *Các ngăn nhớ đứng liền kề nhau thành một dãy liên tục trong bộ nhớ*
  - *Số lượng và kích thước mỗi ngăn là cố định*
  - *Có thể truy nhập trực tiếp vào từng ngăn nhờ chỉ số, nên tốc độ truy nhập vào các ngăn là đồng đều*
- *CTLT động:*
  - *Chiếm các ngăn nhớ thường không liên tục*
  - *Số lượng và kích thước các ngăn có thể thay đổi*
  - *Việc truy nhập trực tiếp vào từng ngăn rất hạn chế, mà thường sử dụng cách truy nhập tuần tự, bắt đầu từ một phần tử đầu, rồi truy nhập lần lượt qua các con trỏ móc nối (liên kết)*

Ngôn ngữ diễn đạt giải thuật

Nguyên tắc khi sử dụng ngôn ngữ:

Có hai nguyên tắc cần lưu ý khi chọn ngôn ngữ diễn đạt giải thuật:

- ***Tính độc lập của giải thuật*** *: ngôn ngữ được chọn phải làm sáng tỏ tinh thần của giải thuật, giúp người đọc dễ dàng hiểu được logic của giải thuật.*

  → Các ngôn ngữ thích hợp là ngôn ngữ tự nhiên và ngôn ngữ hình thức (như các lưu đồ thuật toán, các ký hiệu toán học).

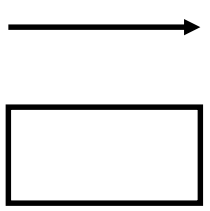
- ***Tính có thể cài đặt được của giải thuật*** *: ngôn ngữ được chọn phải thể hiện được khả năng có thể lập trình được của giải thuật, và giúp người đọc dễ dàng chuyển từ mô tả giải thuật thành chương trình*

  → Các ngôn ngữ lập trình là công cụ tốt nhất vì nó cho ta thấy rõ cài đặt của giải thuật và hoạt động của giải thuật khi chúng ta chạy chương trình trên máy tính

Các loại ngôn ngữ diễn đạt giải thuật

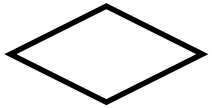
- *Ngôn ngữ tự nhiên*
- *Lưu đồ giải thuật:*
  - *Sử dụng các hình vẽ, biểu tượng để biểu diễn cho các thao tác của giải thuật*
- *Ngôn ngữ lập trình C/C++*

Các thành phần cơ bản của lưu đồ giải thuật



Chỉ đến khối lệnh tiếp theo

Khối lệnh (có thể lệnh đơn hay lệnh phức)



Lệnh rẽ nhánh (điều kiện rẽ nhánh)

Điểm bắt đầu giải thuật

Điểm kết thúc giải thuật

Thiết kế và Phân tích giải thuật

Thiết kế giải thuật

- *Hay nói đúng hơn là thiết kế cấu trúc chương trình mà cài đặt giải thuật. Trong giai đoạn này, chúng ta phải tìm cách biến đổi từ đặc tả giải thuật (mô tả giải thuật làm cái gì, các bước thực hiện những gì) thành một chương trình được viết bằng một ngôn ngữ lập trình cụ thể (giải thuật được cài đặt như thế nào) mà có thể chạy tốt trên máy tính (minh hoạ hoạt động cụ thể của giải thuật).*

Các giai đoạn thiết kế chính

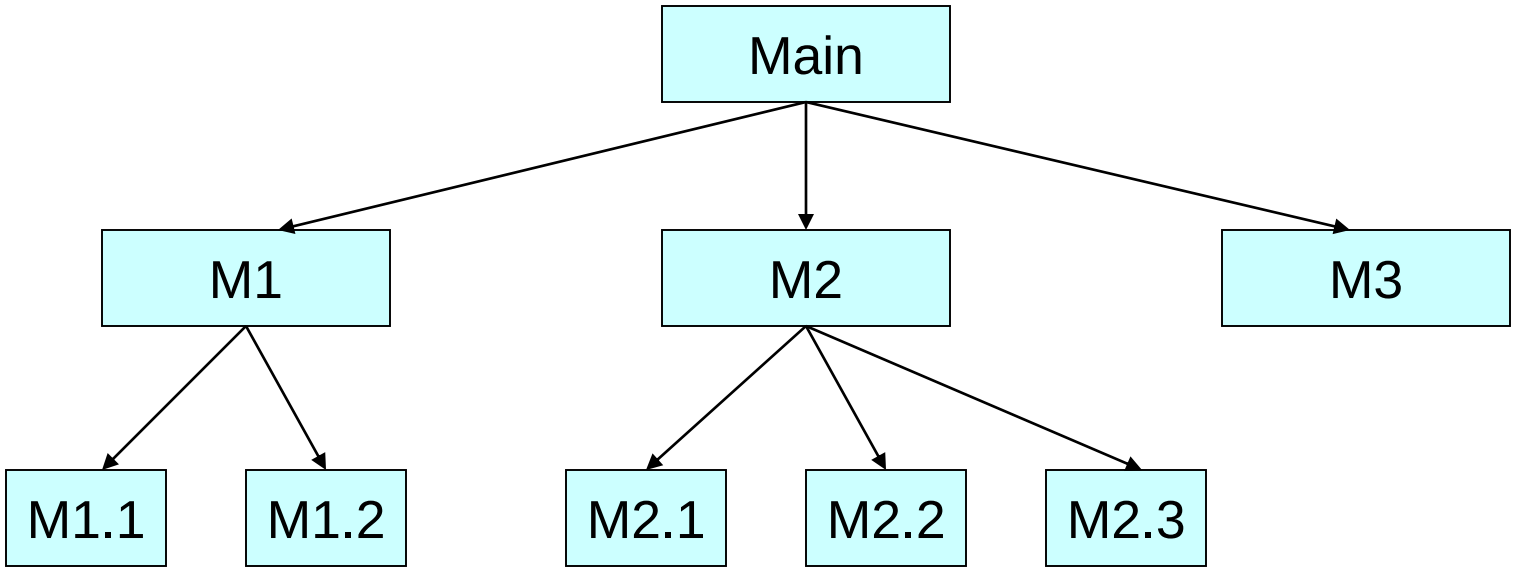
Nói chung, TK thường được chia làm hai giai đoạn chính:

- ***Thiết kế sơ bộ**: đây là giai đoạn cần tìm hiểu cặn kẽ các thành phần của giải thuật. Cụ thể, chúng ta phải biết giải thuật gồm có bao nhiêu thành phần cơ bản, mỗi thành phần đó làm cái gì, giữa các thành phần đó có mối liên quan gì. Mỗi thành phần cơ bản được goi là một mô dul của giải thuật. Phương pháp thiết kế được sử dụng trong giai đoạn này thường là phương pháp **thiết kế từ trên xuống***
- ***Thiết kế chi tiết**: giai đoạn này bắt đầu cài đặt cụ thể các mô dul bằng một ngôn ngữ lập trình cụ thể. Sau đó tiến hành ghép nối các mô dul để tạo thành một chương trình hoàn chỉnh thực hiện giải thuật ban đầu. Phương pháp thiết kế sử dụng trong giai đoạn này thường là phương pháp **tinh chỉnh từng bước***

Phương pháp TK từ trên xuống

- *Còn được gọi khác là **phương pháp mô dul hoá**, nó dựa trên nguyên tắc chia để trị. Chúng ta sẽ chia giải thuật ban đầu thành các giải thuật con (mô dul), mỗi giải thuật con sẽ thực hiện một phần chức năng của giải thuật ban đầu*
- *Quá trình phân chia này được lặp lại cho các modul con cho đến khi các modul là đủ nhỏ để có thể giải trực tiếp*
- *Kết quả phân chia này sẽ tạo ra một **sơ đồ phân cấp chức năng***

Sơ đồ phân cấp chức năng



Phương pháp tinh chỉnh từng bước

- *Phương pháp này chứa các quy tắc cho phép ta thực hiện việc chuyển đổi từ đặc tả giải thuật bằng ngôn ngữ tự nhiên hay lưu đồ sang một đặc tả giải thuật bằng một ngôn ngữ lập trình cụ thể.*
- *Quá trình chuyển đổi này gồm nhiều bước, trong đó mỗi bước là một đặc tả giải thuật.*
- *Trong bước đầu tiên, ta có đặc tả giải thuật bằng ngôn ngữ tự nhiên hay lưu đồ giải thuật. Trong các bước sau, ta tiến hành thay thế dần dần các thành phần được biểu diễn bằng ngôn ngữ tự nhiên của giải thuật bằng các thành phần tương tự được biểu diễn bằng ngôn ngữ lập trình đã chọn. Lặp lại quá trình trên cho đến khi tạo ra một chương trình hoàn chỉnh có thể chạy được, thực hiện giải thuật yêu cầu*

Phân tích giải thuật

Mục đích: Có hai mục đích chính:

- *Tìm hiểu tính đúng đắn của giải thuật để trả lời câu hỏi giải thuật có đúng đắn hay không? Tức là nó cho ra kết quả đúng đối với mọi tập dữ liệu vào hay không.*
- *Tìm hiểu các tài nguyên mà giải thuật sử dụng khi giải thuật được thực hiện trên máy tính, để trả lời câu hỏi giải thuật này chạy như thế nào. Có hai loại tài nguyên chính mà ta quan tâm là thời gian chạy và dung lượng bộ nhớ mà giải thuật cần. Thời gian chạy là một yếu tố căn bản giúp chúng ta đánh giá tính thực tế của giải thuật. Giải thuật luôn phải có thời gian thực hiện hợp lý thì nó mới có tính thực tế, tức là có thể áp dụng được trong các ứng dụng. Nếu không thì nó chỉ có giá trị về mặt lý thuyết.*

Các phương pháp phân tích giải thuật

- *Xác định tính đúng đắn của GT:*
  - *Chứng minh bằng quy nạp*
  - *Chứng minh bằng phản ví dụ*
- *Ước lượng thời gian thực hiện*
  - *Thủ công: dùng đồng hồ đo*
  - *Lý thuyết: xác định độ phức tạp của GT*
- *Ước lượng kích thước bộ nhớ*

Xác định độ phức tạp của GT

- *Khái niệm:*
  - *Quy kết quả tính toán thời gian thực hiện một giải thuật A nào đó về một hàm có dạng $T_A(n)$, với n đại diện cho kích thước dữ liệu vào của giải thuật A (nếu không có gì nhầm lẫn giải thuật thì ta kí hiệu ngắn gọn là T(n)).*

Xác định độ phức tạp của GT

- *Các trường hợp tính T(n):*
  - *T/h tốt nhất* $T_{tn}(n)$
  - *T/h xấu nhất* $T_{xn}(n)$
  - *T/h trung bình* $T_{tb}(n)$

Khái niệm O (ô lớn)

- *Khái niệm O (ô lớn): Cho n là một số nguyên không âm, T(n) và f(n) cũng là các hàm có miền giá trị cũng không âm. Ta nói*

  $$T(n) = O(f(n))$$ *(T(n) là O lớn của f(n))*

  nếu và chỉ nếu tồn tại các hằng số C và n0 sao cho:

  với mọi $n \geq n0$ *thì* $T(n) \leq C.f(n)$
- *Từ định nghĩa ta thấy f(n) là hàm tiệm cận trên của T(n).*

Khái niệm O (ô lớn)

- *Ví dụ: cho $T(n) = 3n$*
  - *Ta có: $T(n) = O(n)$, vì với $C=3$ và $n0=0$, rõ ràng ta có với mọi $n \geq 0$ thì $3n \leq 3.n$.*
  - *Đồng thời ta cũng có $T(n) = O(n^2)$, vì với $n0=3$, $c=1$ ta có với mọi $n \geq 3$ thì $3n \leq 1.n^2$*

Tính chất của O lớn

- *Nếu $T(n) = O(f(n))$ và $f(n) = O(g(n))$*

  ➔ *$T(n) = O(g(n))$.*

- *Nên để biểu diễn độ phức tạp của giải thuật ta luôn chọn f(n) nhỏ nhất và đơn giản nhất sao cho $T(n) = O(f(n))$. Khi đó f(n) được gọi là **hàm độ lớn** hay **độ phức tạp** (hay cấp độ so sánh, hay cấp độ thời gian thực hiện), thường đưa về các dạng sau: 1, $\log_2(n)$, n, $n.\log_2(n)$, n^2, n^3, n^k, 2^n, k^n.*

Các bước xây dựng một CTDL

- *Bước 1: xác định đầy đủ các đặc trưng của CTDL gồm:*
  - *Các thành phần DL có trong CTDL đó,*
  - *Các liên kết (quan hệ) về cấu trúc giữa các thành phần DL.*
- *Bước 2: xác định các thao tác cơ bản trên CTDL: là các thao tác cơ bản, cần thiết nhất để có thể sử dụng được CTDL này.*
- *Bước 3: xác định cấu trúc lưu trữ thích hợp để tổ chức lưu trữ CTDL một cách có hiệu quả. Tính hiệu quả thể hiện ở cả hai mặt: kích thước lưu trữ nhỏ nhất và tốc độ thực hiện các thao tác là nhanh nhất.*

Các bước xây dựng một CTDL

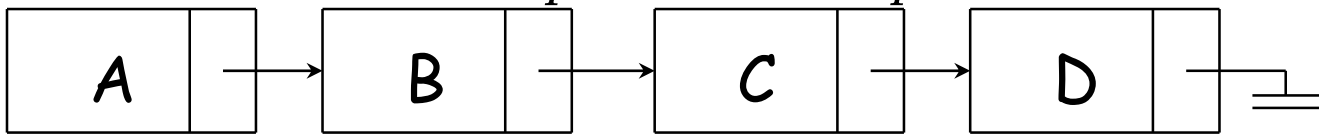
- *<u>Bước 4</u>: Cài đặt các thao tác cơ bản. Việc cài đặt các thao tác phải theo một số nguyên tắc sau:*
  - *Thao tác có khả năng sử dụng lại nhiều lần: sử dụng chương trình con để cài đặt*
  - *Thao tác có tính độc lập về mặt sử dụng và độc lập với các thao tác khác. Để đảm bảo tính chất này thì ta phải chọn các tham số hợp lí cho các thao tác*
  - *Thao tác phải hiệu quả*

3. 4 Một số cấu trúc dữ liệu

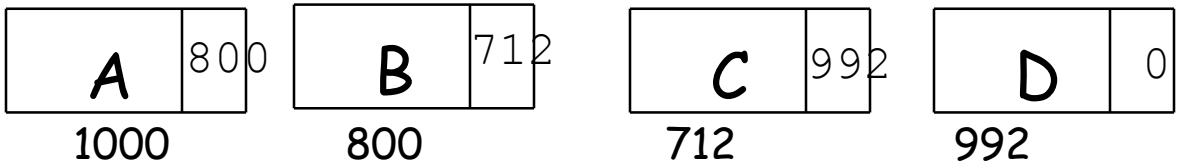
- *Cấu trúc tuyến tính :*
  - *Mảng*
  - *Danh sách: LIFO (Stack_ngăn xếp), FIFO (Queue_hàng đợi)*
  - *Danh sách móc nối (Danh sách liên kết)*
- *Cấu trúc phi tuyến:*
  - *Cây*
  - *Đồ thị*

Danh sách liên kết đơn

- *Là tập hợp các phần tử dữ liệu không liên tục được kết nối với nhau thông qua một liên kết (thường là con trỏ)*
- *Cho phép ta quản lý bộ nhớ linh động*
- *Các phần tử được chèn vào DS và xóa khỏi DS một cách dễ dàng*
- *Tại mỗi nút có hai thành phần:*
  - *Dữ liệu trong nút*
  - *Con trỏ trỏ đến phần tử kế tiếp*



Trong bộ nhớ



Danh sách liên kết đơn(...)

- *Cú pháp khai báo một phần tử của danh sách liên kết trong C:*

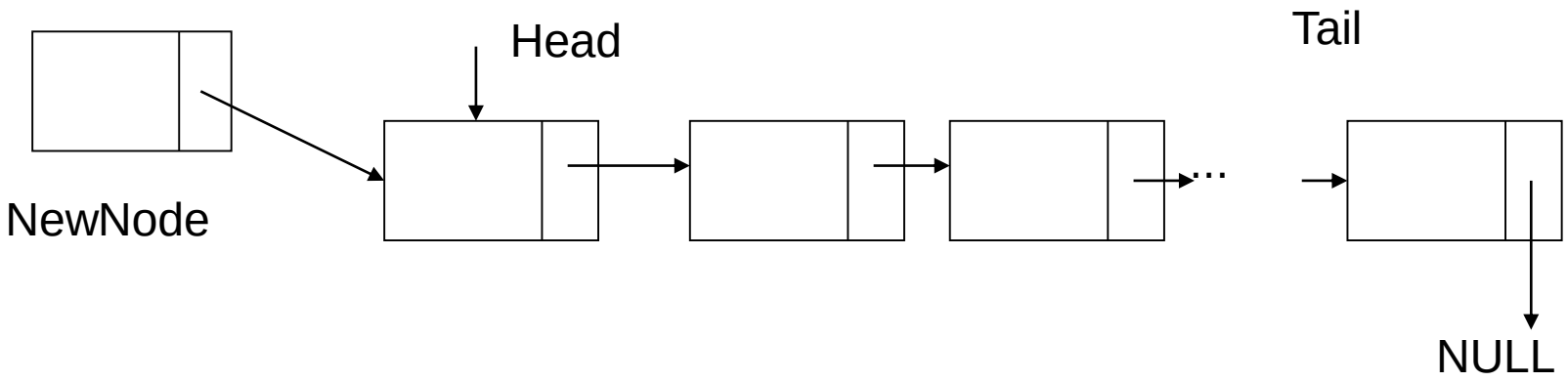
```
struct Node
        {
                Kieu_PT data;
                Node *next;
        };
```

Giải thuật thêm một phần tử vào danh sách liên kết

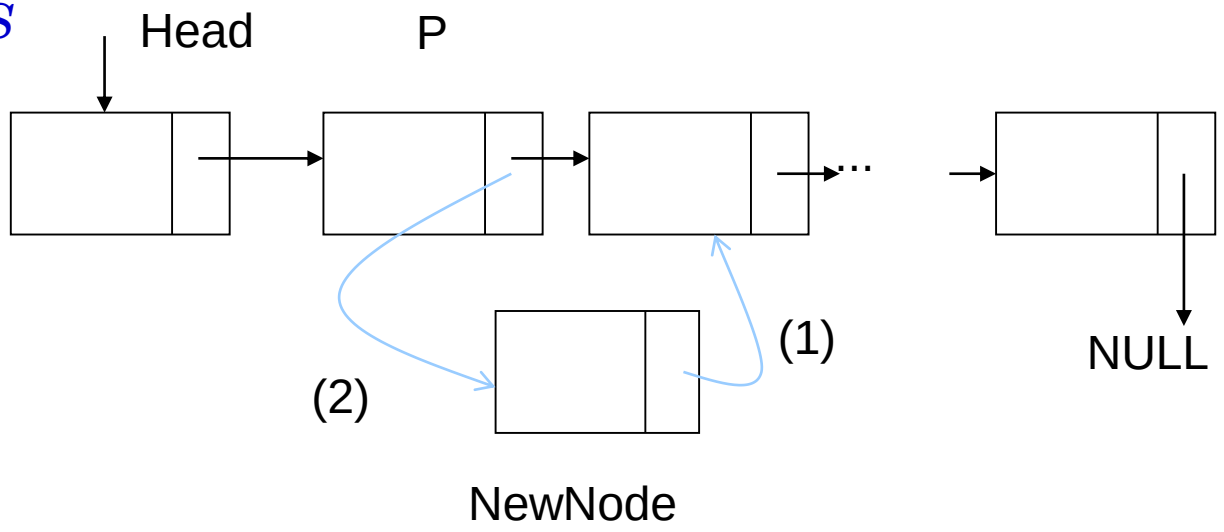
- *Cấp phát bộ nhớ cho node*
- *Đưa số liệu vào node mới*
- *Cho node liên kết với danh sách:*
  - *Thêm vào đầu DS*
  - *Thêm vào cuối DS*
  - *Chèn vào giữa DS*

Danh sách liên kết đơn(...)

Thêm vào đầu DS



Thêm vào giữa DS sau con trỏ P



Danh sách liên kết đơn (...)

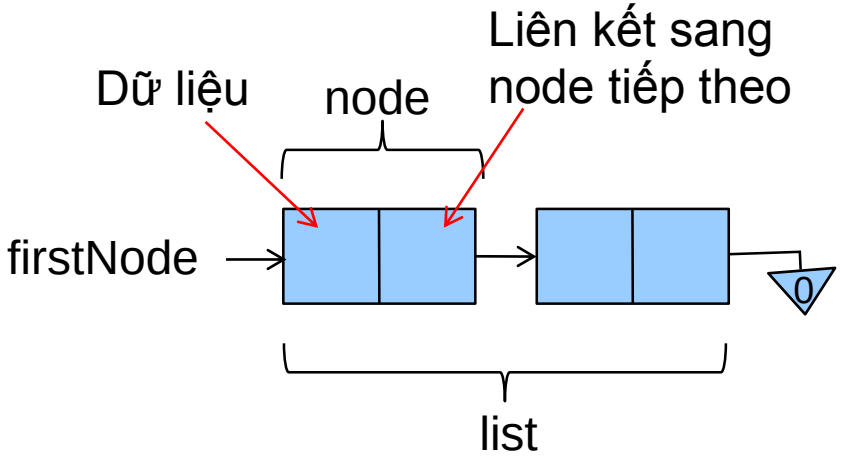
- *Thuật toán để duyệt một DS liên kết:*

```
while(DS !=NULL)
{
        ... Các thao tác với dữ liệu trên Node
        node=node->next; /* chuyển qua node tiếp theo
        */
}
```

Ví dụ:

- *Một ví dụ đơn giản minh họa về cấu trúc list có liên kết một chiều:*
  - *Mỗi node trong list chỉ chứa một dữ liệu thuộc kiểu int.*
  - *Hỗ trợ các hàm khởi tạo list, bổ sung phần tử, xóa phần tử đầu list, xóa phần tử sau một phần tử nào đó, bổ sung phần tử sau phần tử nào đó, xóa tất cả các phần tử.*

```
struct Node{
        int data;
        Node* next;
};

struct List{
        Node* firstNode;
};
```



- *Các khai báo hàm*

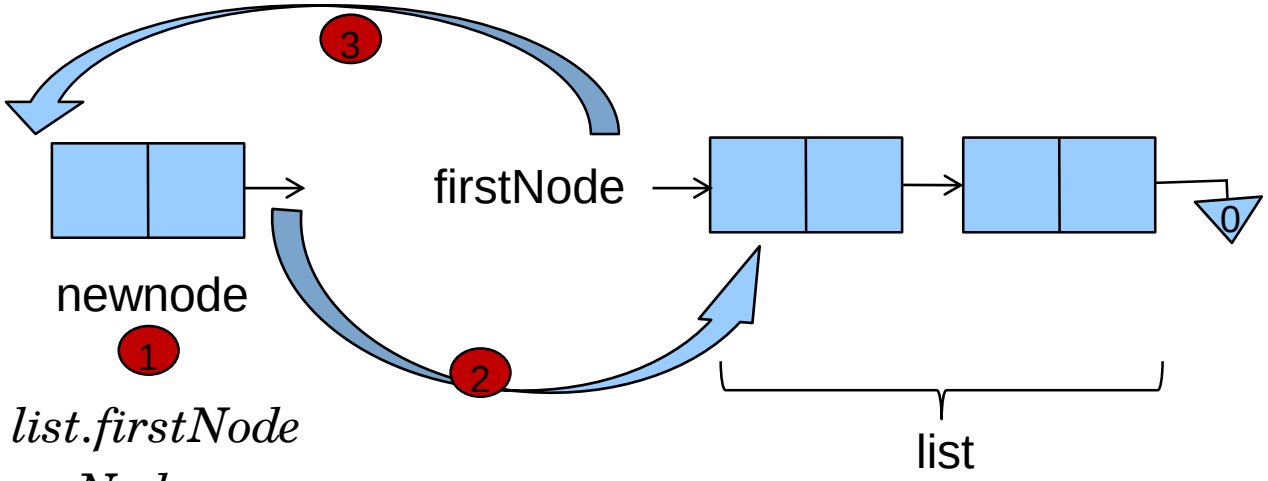
```
void List_Init(List* list);
void List_Add(List* l, int data);
void List_InsertAfter(Node* node, int data);
void List_RemoveFirst(List* list);
void List_RemoveAfter(Node* node);
void List_DeleteAll(List* list);
int List_Length(List* list);
void List_Display(List* list)
Node* List_Search(List* list, int data);
```

- ***Khởi tạo List: gán con trỏ fistnode = 0***

```
void List_Init(List* list){
        list->firstNode = 0;
}
```

- ***Bổ sung phần tử vào đầu của List:***

Giải thuật:



B2: newNode.next := list.firstNode

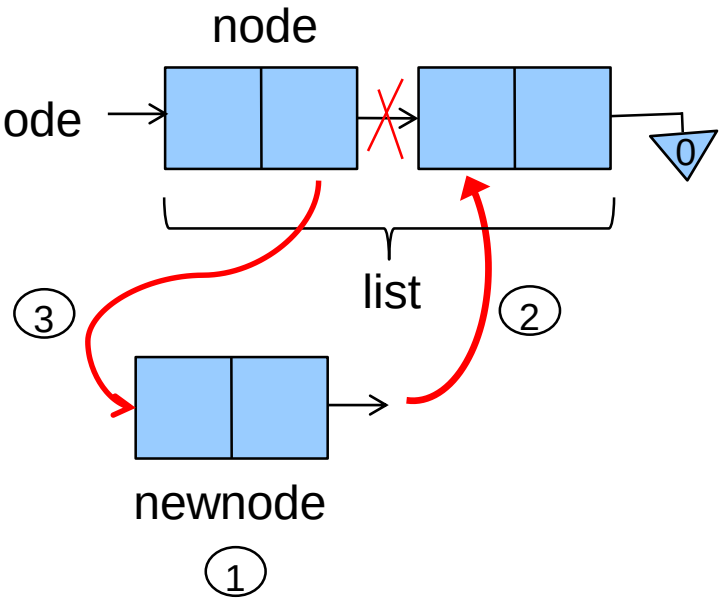
B3: list.firstNode := newNode

```
void List_Add(List* list, int data){
        Node* newnode = new Node;
        newnode->data = data;
        newnode->next = list->firstNode;
        list->firstNode = newnode;
}
```

- ***Bổ sung vào giữa list, sau một phần tử nào đó***

Giải thuật:

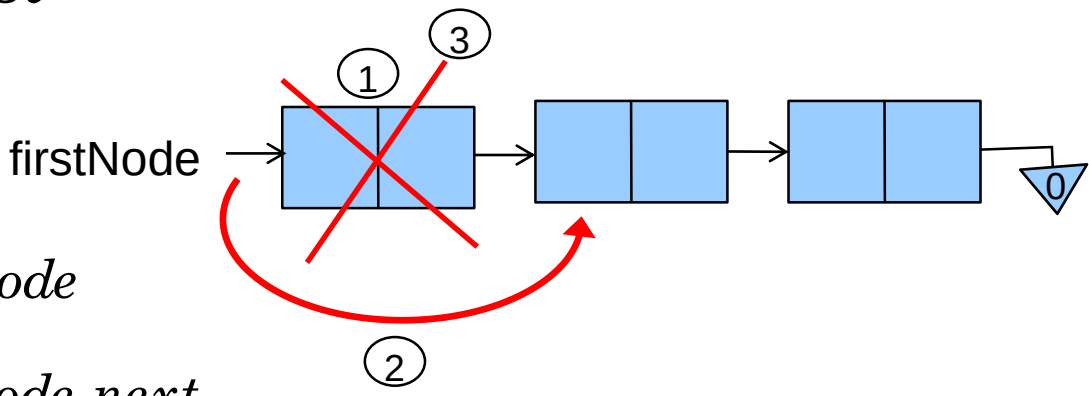
- *B2: newNode.next := node.next*
- *B3: node.next := newNode*




```
void List_InsertAfter(Node* node, int data){
        Node* newnode = new Node;
        newnode->data = data;
        newnode->next = node->next;
        node->next = newnode;
}
```

- ***Xóa phần tử đầu list***

Giải thuật:



B1: obsoleteNode := list.firstNode

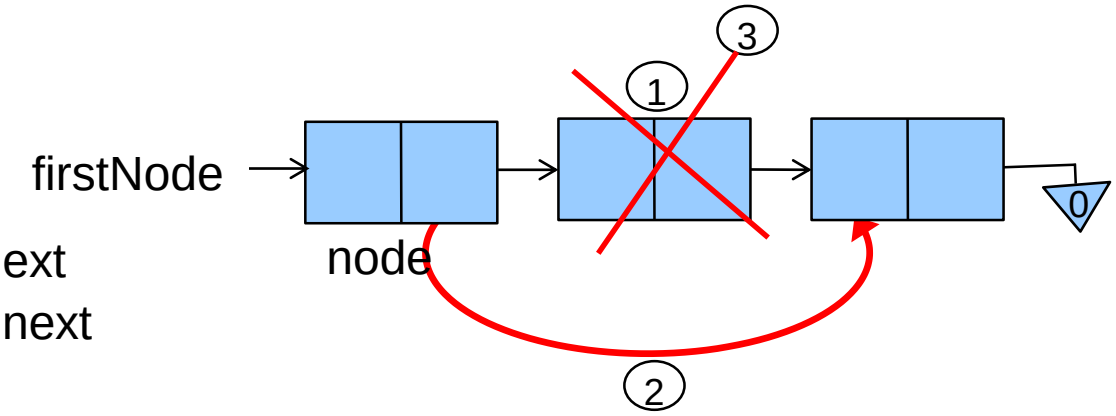
B2: list.firstNode := list.firstNode.next

B3: destroy obsoleteNode

```
void List_RemoveFirst(List* list){
        Node* obsoleteNode = list->firstNode;
        list->firstNode = list->firstNode->next;
        delete obsoleteNode ;
}
```

- ***Xóa phần tử sau một phần tử nào đó***

Giải thuật:



- obsoleteNode := node.next
- node.next := node.next.next

- destroy obsoleteNode

```
void List_RemoveAfter(Node* node){
        Node* obsoleteNode = node->next;
        node->next = obsoleteNode ->next;
        delete obsoleteNode ;
}
```

- ***Xóa tất cả các phần tử của list***

Giải thuật:

```
node := list.firstNode
while node not null {
        (…delete node)
        node := node.next
}
```

```
void List_DeleteAll(List* list){
        Node* node = list->firstNode;
        while (node != 0){
                Node* nextnode = node->next;
                delete node;
                node = nextnode;
        }
        list->firstNode = 0;
}
```

- ***Xác định độ dài của list (số phần tử của list)***

```
int List_Length(List* list){
        Node* node = list->firstNode;
        int i = 0;
        while(node != 0){
                i++;
                node = node->next;
        }
        return i;
}
```

- *Để hỗ trợ kiểm tra kết quả, thực hiện một hàm hiển thị như sau:*

```
void List_Display(List* list){
	Node* node = list->firstNode;
	int i = List_Length(list);
	cout<<"\nDo dai cua list:\t"<<i;
	if (list->firstNode == 0)
		cout<<"\nList rong\r\n";
	else
	{
		while(node != 0){
			cout<<"\ndia chi cua node "<< i<<"\t"<<&node->data;
			cout<<"\nnode->data:\t\t"<<node->data;
			cout<<"\nnode->next:\t\t"<<node->next<<"\n";
			node = node->next;
			i--;
		}
		cout<<endl;
	}
}
```

- ***Hàm tìm kiếm một phần tử:***
- *Mục đích: trả về một phần tử có dữ liệu bằng dữ liệu cho trước*

```
Node* List_Search(List* list, int data){
        Node* node = list->firstNode;

        while (node != 0){
                if(node->data == data)
                        return node;
                else
                        node = node->next;
        }
        return 0;
}
```

- ***Sử dụng:***

```
void main(){
        List l;

        List_Init(&l);
        List_Display(&l);

        List_Add(&l, 1);//list: 1
        List_Add(&l, 2);//list: 2 1
        cout<<"Noi dung List sau hai lenh List_Add:";
        List_Display(&l);

        cout<<"Noi dung List sau khi them 2 lenh List_Add nua:";
        List_Add(&l, 3);//list: 3 2 1
        List_Add(&l, 4);//list: 4 3 2 1
        List_Display(&l);
```

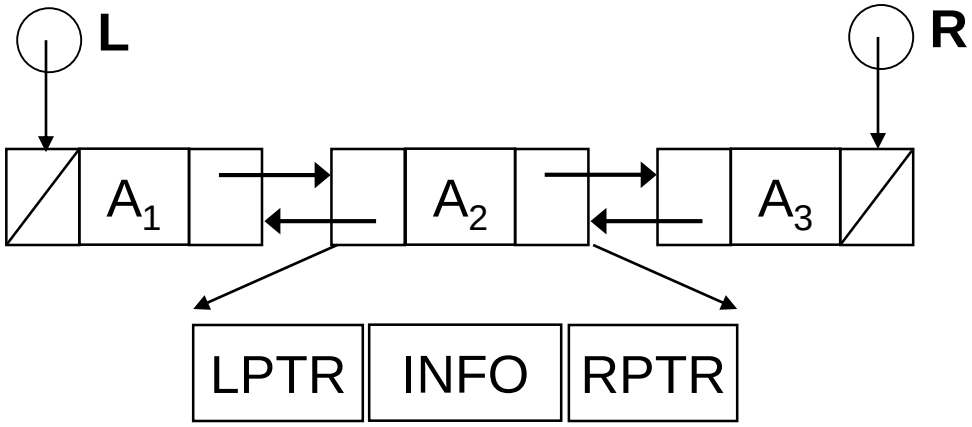
```
        cout<<"Noi dung sau khi removefirst:";
        List_RemoveFirst(&l);
        List_Display(&l);

        cout<<"Noi dung sau khi remove node dung sau node co data = 3:";
        List_RemoveAfter(List_Search(&l,3));
        List_Display(&l);

        cout<<"Noi dung sau khi chen node dung sau node co data = 3:";
        List_InsertAfter(List_Search(&l,3),100);
        List_Display(&l);

        cout<<"Noi dung sau khi xoa tat ca:";
        List_DeleteAll(&l);
        List_Display(&l);

}
```

- ***Kết quả trên mà hình***

```
Do dai cua list:          0
List rong
Noi dung List sau hai lenh addNode:
Do dai cua list:          2
dia chi cua node 2        0x00431810
node->data:               2
node->next:               0x00431850

dia chi cua node 1        0x00431850
node->data:               1
node->next:               0x00000000

Noi dung List sau khi them 2 lenh addNode nua:
Do dai cua list:          4
dia chi cua node 4        0x00431790
node->data:               4
node->next:               0x004317D0

dia chi cua node 3        0x004317D0
node->data:               3
node->next:               0x00431810

dia chi cua node 2        0x00431810
node->data:               2
node->next:               0x00431850

dia chi cua node 1        0x00431850
node->data:               1
node->next:               0x00000000
```

```
Noi dung sau khi removefirst:
Do dai cua list:        3
dia chi cua node 3      0x004317D0
node->data:             3
node->next:             0x00431810

dia chi cua node 2      0x00431810
node->data:             2
node->next:             0x00431850

dia chi cua node 1      0x00431850
node->data:             1
node->next:             0x00000000

Noi dung sau khi remove node dung sau node co da
Do dai cua list:        2
dia chi cua node 2      0x004317D0
node->data:             3
node->next:             0x00431850

dia chi cua node 1      0x00431850
node->data:             1
node->next:             0x00000000

Noi dung sau khi chen node dung sau node co data
Do dai cua list:        3
dia chi cua node 3      0x004317D0
node->data:             3
node->next:             0x00431810

dia chi cua node 2      0x00431810
node->data:             100
node->next:             0x00431850

dia chi cua node 1      0x00431850
node->data:             1
node->next:             0x00000000

Noi dung sau khi xoa tat ca:
Do dai cua list:        0
List rong
```

Danh sách liên kết đôi



- *Sử dụng 2 con trỏ, giúp ta luôn xem xét được cả 2 chiều của danh sách*
- *Tốn bộ nhớ nhiều hơn*
- *Con trỏ trái (LPTR): trỏ tới thành phần bên trái (phía trước)*
- *Con trỏ phải (RPTR): trỏ tới thành phần bên phải (phía sau)*

Bài tập về nhà

Thực hiện một cấu trúc list quản lý sinh viên. Thông tin về sinh viên được biểu diễn bởi một struct gồm mã sinh viên (int), họ tên sinh viên (string), lớp (string).

Chương trình có chức năng:

-Nhập thông tin sinh viên từ bàn phím

-Thêm sinh viên vào cuối danh sách

-Hiển thị danh sách sinh viên

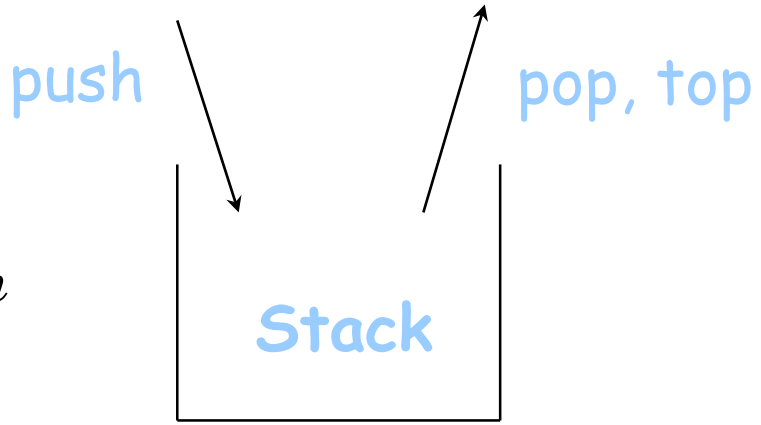
-Tìm sinh viên theo tên

-Xóa sinh viên theo tên

Viết hàm main minh họa cách sử dụng

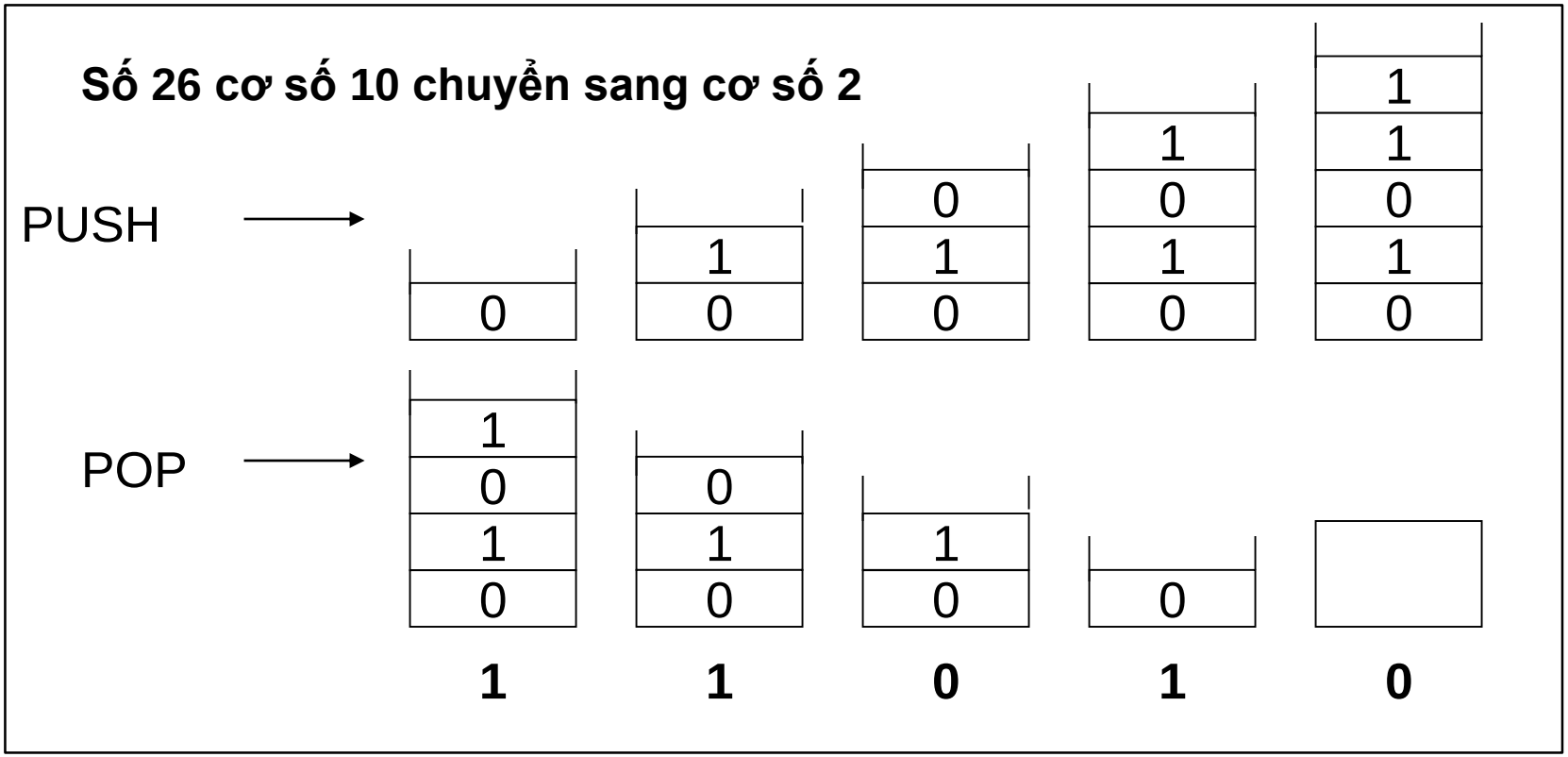
Ngăn xếp (Stack-LIFO)

- *Là tập hợp các phần tử có trình tự, các phần tử vào/ra ngăn xếp gọi là "đỉnh ngăn xếp" (stack-top)*
- *Các phần tử vào ra theo nguyên tắc LIFO (Last In First Out)*
- *Các thao tác cơ bản: push, pop, con trỏ top, hàm khởi tạo stack initialize(), hàm kiểm tra stack có rỗng hay không isEmpty()*
- *Để xây dựng ngăn xếp có thể dùng:*
  - *Mảng kết hợp với một biến nguyên "top" lưu vị trí trên cùng của ngăn xếp*
  - *Danh sách liên kết với thuật toán chèn/xóa ở đầu của danh sách.*



Ứng dung của stack

- *Đổi cơ số: VD từ 10 sang 2*
- *Cách làm: ta chia liên tiếp cho 2 và lấy các số dư theo chiều ngược lại*



Ứng dung của stack (...)

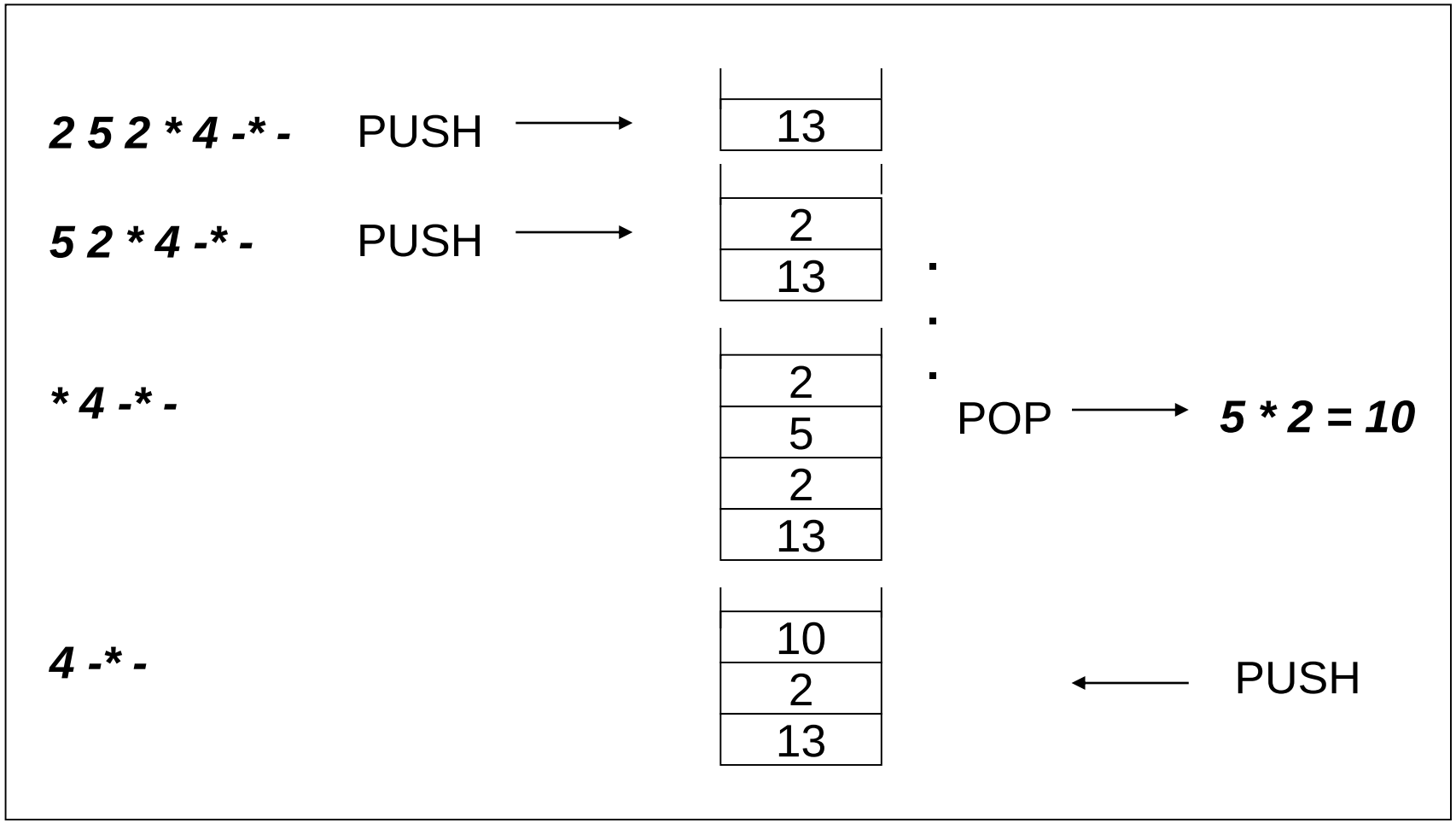
- *Định giá trị biểu thức theo phương pháp nghịch đảo Balan*

Ví dụ: tính biểu thức 13 – 2 * (5 * 2 – 4)

- *Khái niệm **infix, postfix, prefix***
  - *infix:3 + 4* ***postfix: 3 4 +*** *prefix: + 3 4*
  - *infix:2 + 3 * 4* ***postfix: 2 3 4 * +*** *prefix: + 2 * 3 4*
  - *infix:2 * 3 + 4* ***postfix: 2 3 * 4 +*** *prefix: + * 2 3 4*
- *Ứng dụng hậu tố (**postfix**) vào ví dụ*
  - ***13 2 5 2 * 4 -* -***
  - *Đọc từ trái qua phải*
  - *Nếu là số: PUSH vào Stack*
  - *Nếu là phép tính (operator) thì lấy số (POP) trong Stack ra, tính toán rồi lại PUSH vào*

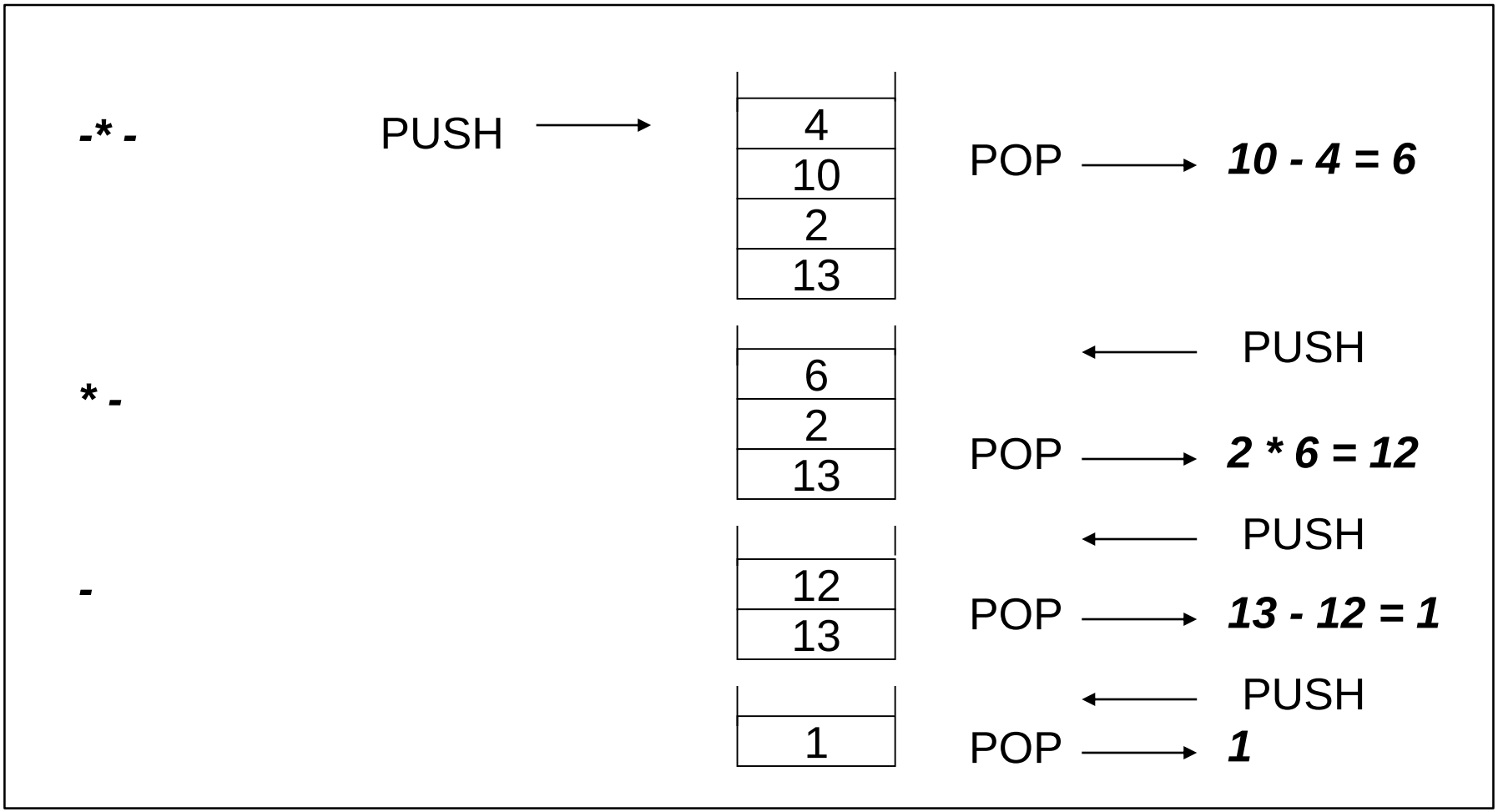
Ứng dung của stack (...)

- ***Ví dụ: tính biểu thức 13 – 2 * (5 * 2 – 4)***
- ***Biểu diễn dưới dạng hậu tố: 13 2 5 2 * 4 - * -***



Ứng dung của stack (...)

- ***Ví dụ: tính biểu thức 13 – 2 * (5 * 2 – 4) (tiếp...)***



Ví dụ: Đổi cơ số dùng stack

```
#include <iostream>
using namespace std;
struct Node
{
        int data;
        Node *next;
};
struct Stack
{
        Node *top;
};
void Khoi_Tao(Stack &s)
{
        s.top =NULL;
}
```

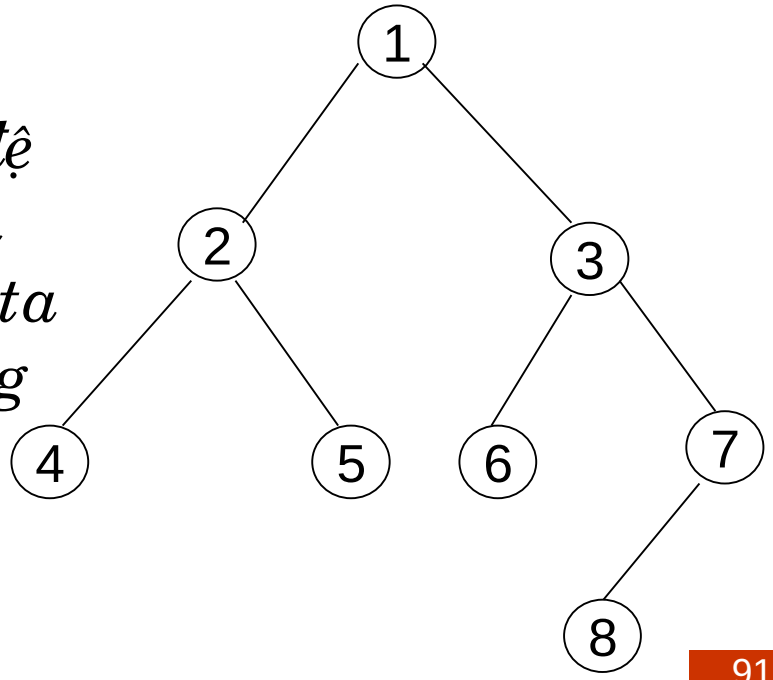
```
void Push(Stack &s,int pt)
{
    Node *tmp= new Node;
    tmp->data =pt;
    if(s.top ==NULL) /* Phan tu dau tien */
    {
            s.top =tmp;
            s.top->next =NULL;
    }
    else  /* Phan tu tiep theo */
    {
            tmp->next =s.top;
            s.top =tmp;
    }

}
```

```
bool isEmpty(Stack s)
{
        return (s.top==NULL);
}
void Pop(Stack &s,int &pt)
{
        if (!isEmpty(s))
        {
                pt=s.top->data ;
                s.top=s.top->next ;
        }
}
```

```
void main() /* Quy doi co so 10 ra nhi phan */
{
        Stack S;
        Khoi_Tao(S);
        int a,pt;
        cout<<"Nhap vao so he 10:\n";
        cin>>a;
        while((a/2)>0)
        {
                Push(S,a%2);
                a=a/2;
        }
        Push(S,a);
        cout<<"Bieu dien duoi dang nhi phan:\n";
        while(!isEmpty(S))
        {
                Pop(S,pt);
                cout<<pt;
        }
        cout<<"\n";
}
```

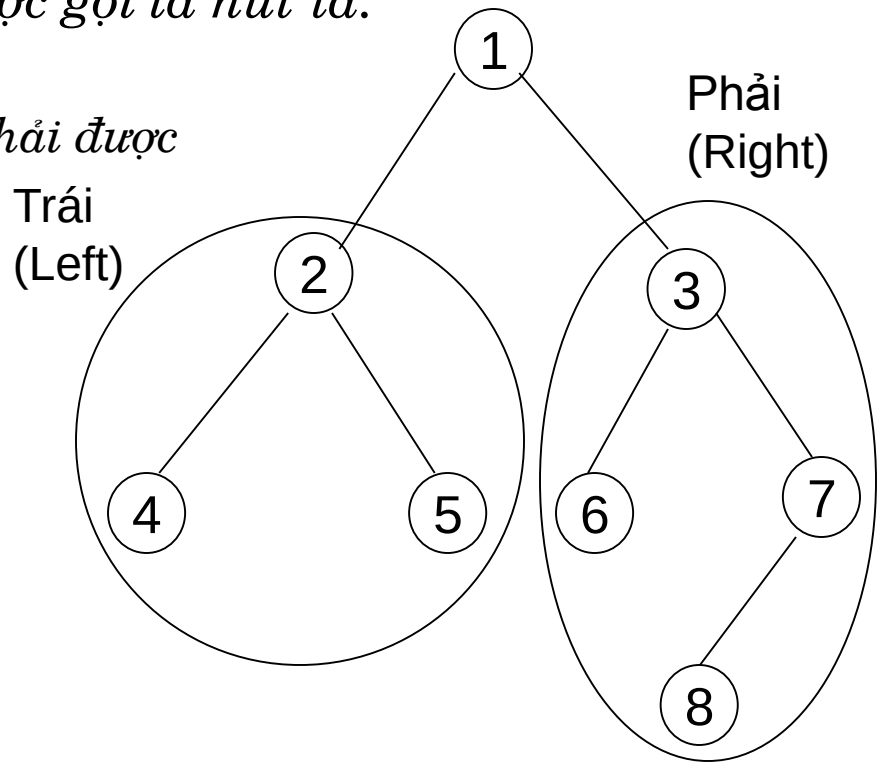
Cây nhị phân (Binary Tree)

- *Mô tả cây nhị phân và các khái niệm cơ bản*
  - *Cây có thứ tự và là cây cấp hai, tức là mỗi nút có tối đa 2 con.*
  - *Hai con của một nút được phân biệt thứ tự và quy ước nút trước gọi là nút con trái và nút sau được gọi là nút con phải*
- *Khi biểu diễn cây nhị phân, ta cũng cần có sự phân biệt rõ ràng giữa con trái và con phải, nhất là khi nút chỉ có một con.*
- *Cây nhị phân có tính chất đệ quy: nếu ta chặt một nhánh bất kì của cây nhị phân thì ta sẽ thu được hai cây con cũng đều là cây nhị phân.*



Cây nhị phân (...)

- *Mô tả cây nhị phân và các khái niệm cơ bản*
  - *Loại nút:*
    - *Nút có đủ hai con được gọi là nút kép: 2, 3*
    - *Nút chỉ có một con gọi là nút đơn: 7*
    - *Nút không có con vẫn được gọi là nút lá: 4, 5, 6, 8*
  - *Cây con có gốc là nút con trái/phải được gọi là cây con trái/phải.*



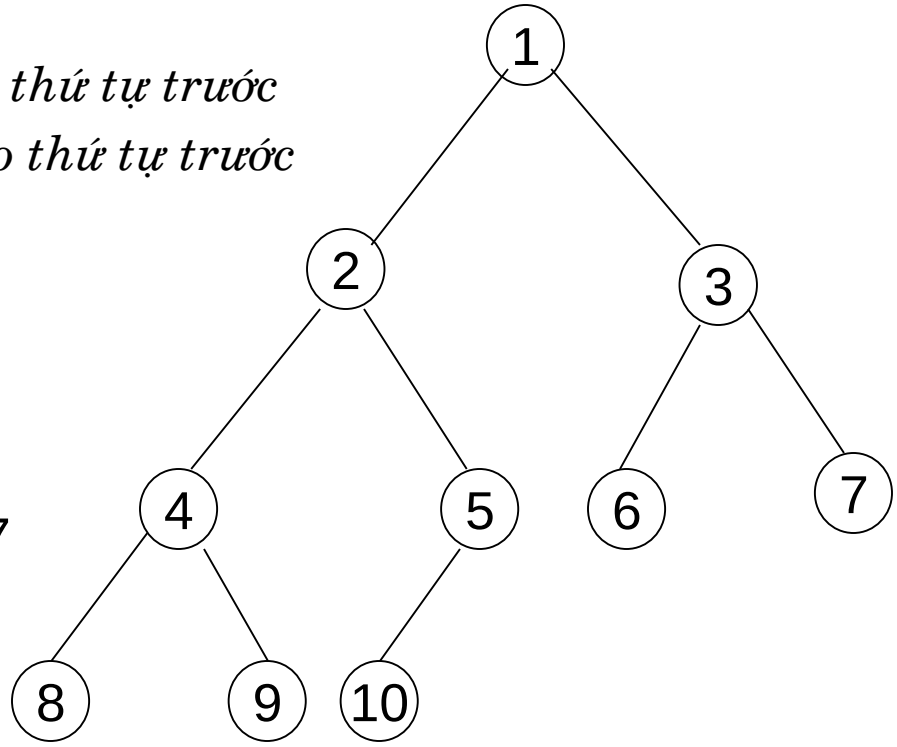
Cây nhị phân (...)

- *Các thao tác cơ bản - Giải thuật duyệt cây*
  - *Phép duyệt cây*
    - *Phép thăm một nút (visit): là thao tác truy nhập vào một nút của cây để xử lý nút đó.*
    - *Phép duyệt (traversal): là phép thăm một cách hệ thống tất cả các nút của cây, mỗi nút đúng một lần.*
- *Có 3 phép duyệt cây thông dụng:*
  - *Duyệt theo thứ tự trước (preorder traversal)*
  - *Duyệt theo thứ tự giữa (inorder traversal)*
  - *Duyệt theo thứ tự sau (postorder traversal)*

Cây nhị phân (...)

- *Các thao tác cơ bản - Giải thuật duyệt cây*
  - *Giải thuật duyệt theo thứ tự trước (preorder traversal, còn gọi là duyệt cây theo chiều sâu): đây là giải thuật đệ quy*
    - *Trường hợp đệ quy: để duyệt cây nhị phân T, ta theo các bước sau:*
      - *Thăm gốc*
      - *Duyệt cây con trái theo thứ tự trước*
      - *Duyệt cây con phải theo thứ tự trước*
    - *Trường hợp điểm dừng:*
      - *Khi cây T rỗng.*

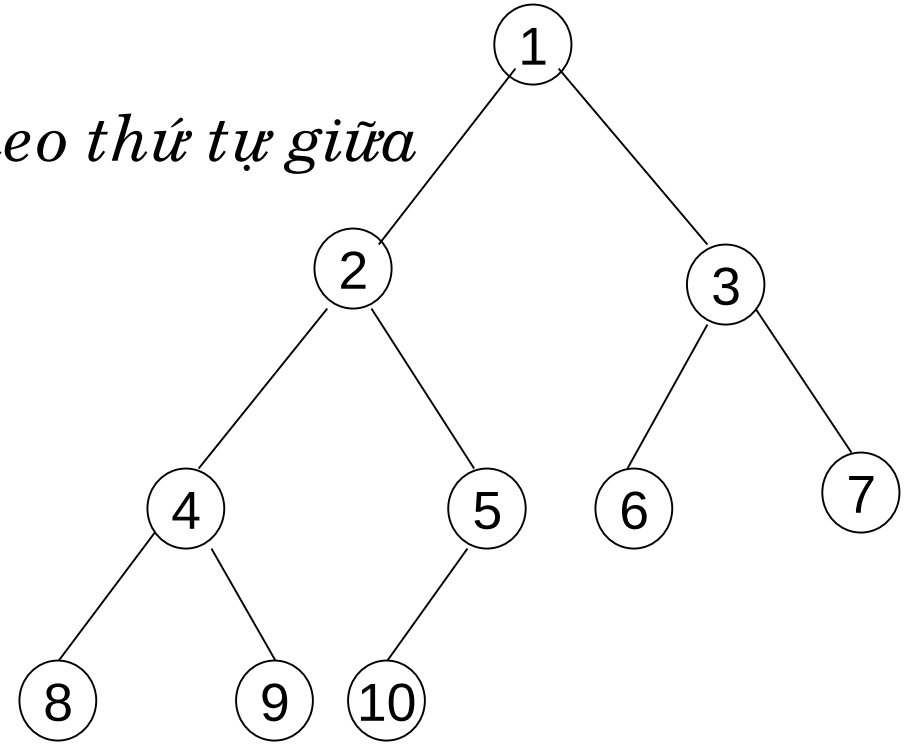
VD: 1, 2, 4, 8, 9, 5, 10, 3, 6, 7



Giải thuật duyệt cây (...)

- *Giải thuật duyệt theo thứ tự giữa (inorder travelsal): đây là giải thuật đệ quy.*
  - *Trường hợp đệ quy: để duyệt cây nhị phân T, ta theo các bước sau:*
    - *Duyệt cây con trái theo thứ tự giữa*
    - *Thăm gốc*
    - *Duyệt cây con phải theo thứ tự giữa*
  - *Trường hợp điểm dừng:*
    - *Khi cây T rỗng.*

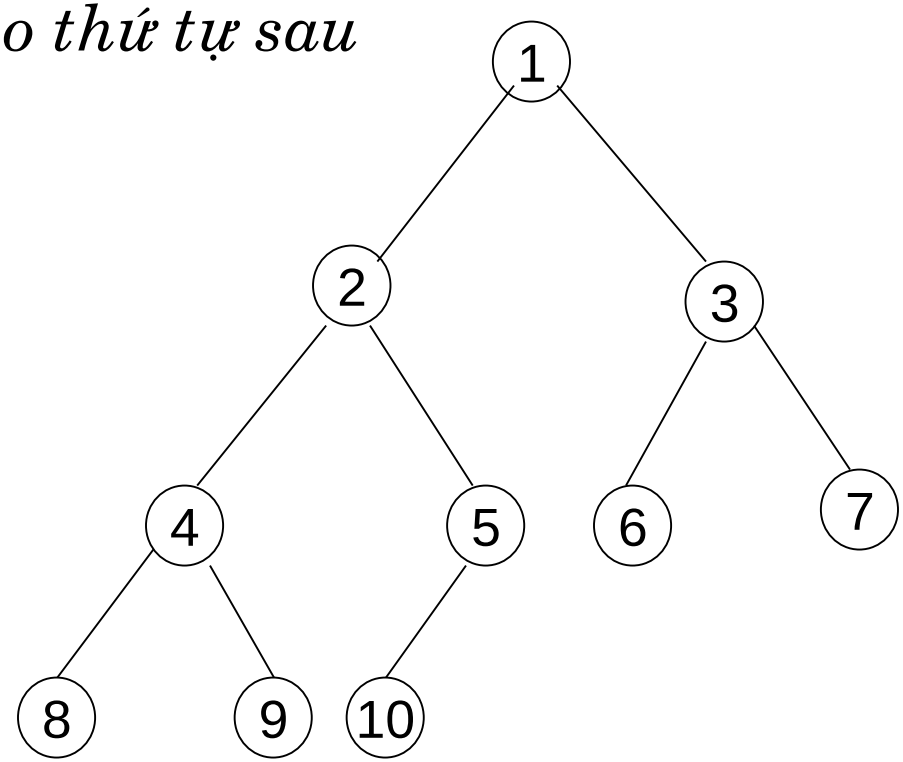
VD: 8, 4, 9, 2, 10, 5, 1, 6, 3, 7



Giải thuật duyệt cây (...)

- *Giải thuật duyệt theo thứ tự sau (postorder travelsal): đây là giải thuật đệ quy.*
  - *Trường hợp đệ quy: để duyệt cây nhị phân T, ta theo các bước sau:*
    - *Duyệt cây con trái theo thứ tự sau*
    - *Duyệt cây con phải theo thứ tự sau*
    - *Thăm gốc*
  - *Trường hợp điểm dừng:*
    - *Khi cây T rỗng.*

VD: 8, 9, 4, 10, 5, 2, 6, 7, 3, 1

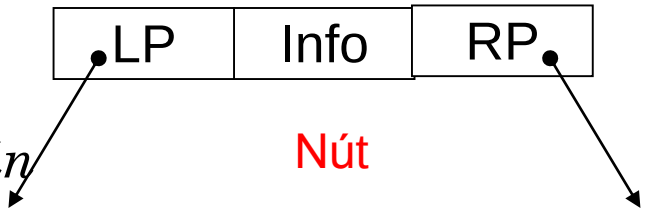


Cài đặt cây nhị phân

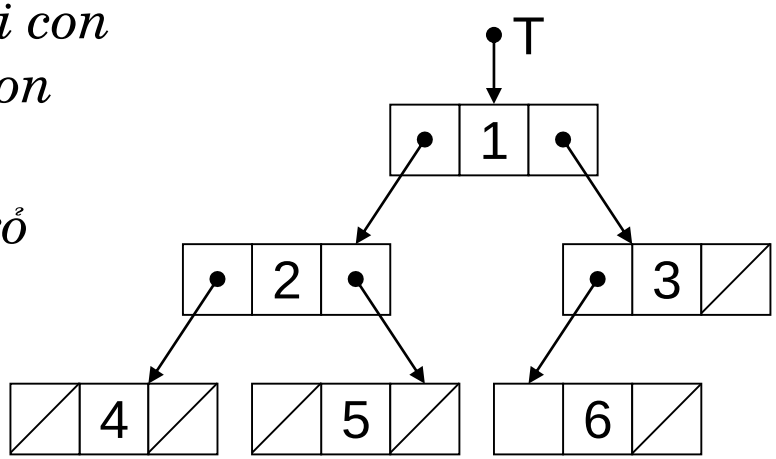
- *Nguyên tắc cài đặt: Sử dụng cấu trúc lưu trữ móc nối*
  - *Đối với cây nhị phân, ta cần lưu trữ hai thành phần:*
  - *Các phần tử (nút) của cấu trúc cây:*
    - *Dùng các nút của CTLT để lưu trữ các phần tử.*
  - *Các nhánh (quan hệ cha-con) giữa các cặp nút:*
    - *Sử dụng con trỏ để biểu diễn các nhánh.*
    - *Do mỗi nút có nhiều nhất hai con nên trong mỗi nút của CTLT ta sẽ có hai con trỏ để trỏ đến tối đa hai con này.*

Cài đặt cây nhị phân (...)

- *Dùng cấu trúc móc nối*
  - *Cấu tạo mỗi nút (xem hình):*
    - *Trường Info: chứa thông tin về một phần tử.*
    - *Trường LP và RP tương ứng là hai con trỏ sẽ trỏ vào hai nút con trái và con phải.*
    - *Nếu không có con thì giá trị con trỏ tương ứng sẽ rỗng (NIL/NULL)*
    - *Khai báo cấu trúc một nút:*
  - *Tổ chức cấu trúc cây như sau (xem hình):*
    - *Ít nhất một con trỏ T trỏ vào nút gốc, vừa đại diện cho cây, vừa là điểm truy nhập vào các nút khác trong cây.*
    - *Các thao tác muốn truy nhập vào các nút trong cây phải thông qua con trỏ này.*



Nút



Cây

Danh sách liên kết sinh viên

```
#include "stdafx.h"
#include <iostream>
#include <string>
using namespace std;
struct sv{
        int id;
        string name;
        string clas;
        sv* next;
};
```

```
struct ds{
        sv* firstsv;
};
void dssv_Init(ds* dssv){
        dssv->firstsv = 0;
}
void nhap_sv(int &id, string & name, string &clas)
{       cout<<"nhap ma sinh vien: ";cin >>id;
        cout<<"\n nhap ho ten: ";
        //cin.ignore (1); getline(cin,name); //cin.ignore (1);
        cin>>name;
        cout<<"\n nhap lop: ";
        //getline(cin,clas); //cin.ignore (1);
        cin>>clas;
}
```

```
void them_sv(ds* dssv, int id, string  name, string  clas)
{

        sv* newsv = new sv;
        newsv->id = id;
        newsv->name =name;
        newsv->clas =clas;
        newsv->next = dssv->firstsv;
        dssv->firstsv = newsv;

}
```

```
void them_sv_Final_nhap(ds* dssv)
{
        int id;
        string name;
        string clas;
        nhap_sv(id, name, clas);
        sv* newsv = new sv;
        newsv->id = id;
        newsv->name =name;
        newsv->clas =clas;
        sv* node = dssv->firstsv;
                while(node->next != 0)
        {
                node = node->next;
        }

        newsv->next = node->next;
        node->next=newsv;
}
```

```
void them_sv_Final(ds* dssv, int id, string name, string clas)
{
        sv* newsv = new sv;
        newsv->id = id;
        newsv->name =name;
        newsv->clas =clas;
        sv* node = dssv->firstsv;
        while(node->next != 0)
        {
                node = node->next;
        }

        newsv->next = node->next;
        node->next=newsv;
}
```

```
int ds_Length(ds* dssv){
        sv* node = dssv->firstsv;
        int i = 0;
        while(node != 0)
        {
                i++;
                node = node->next;
        }
        return i;
}
```

```
void dssv_Display(ds* dssv){
	sv* node = dssv->firstsv;
	int i = ds_Length(dssv);
	cout<<"\nDo dai cua list:\t"<<i;
	if (dssv->firstsv == 0)
		cout<<"\nList rong\n\n";
	else
	{
		while(node != 0){
			cout<<"\nsinh vien thu "<< i<<"\n";
			cout<<"\nid:\t\t"<<node->id;
			cout<<"\nname:\t\t"<<node->name;
			cout<<"\nclass:\t\t"<<node->clas;
			//cout<<"\nnode->next:\t\t"<<node->next<<"\n";
			node = node->next;
			i--;
		}
		cout<<endl;
	}
}
```

```
sv* dssv_Search(ds* dssv, string name){
        sv* node = dssv->firstsv;

        while (node != 0){
                if(node->name == name)
                        return node;
                else
                        node = node->next;
        }
        return 0;
}
```

```
void dssv_remove(ds* dssv, string name)
{
        if (dssv->firstsv ->name==name)
        {
                sv* obsoleteSv=dssv->firstsv ;
                dssv->firstsv =dssv->firstsv ->next ;
                delete obsoleteSv;
        }
        else
        {
                sv* node=dssv->firstsv ;
                while (node->next!=0)
                {
                        if(node->next ->name ==name)
                        {
                                sv* obsoleteSv=node->next ;
                                node->next =node->next->next ;
                                delete obsoleteSv;
                        }
node=node->next ;
                }
        }
}
```

```
void main(){
	ds dssv;

	dssv_Init(&dssv);

	them_sv(&dssv, 01, "nguyen van a", "dt5");//list: 1
	//them_sv_Final(&dssv, 02, "nguyen van b", "dt6");
	//dssv_Display(&dssv);
	them_sv_Final_nhap(&dssv);
	//dssv_remove(&dssv, "nguyen van a");
	//cout<<"Noi dung List sau hai lenh List_Add:";
	dssv_Display(&dssv);



}
```