UBND TỈNH LÂM ĐỒNG
TRƯỜNG CAO ĐẲNG ĐÀ LẠT

GIÁO TRÌNH
MÔN HỌC: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

NGÀNH/NGHỀ: CÔNG NGHỆ THÔNG TIN (ỨNG DỤNG PHẦN MỀM)

TRÌNH ĐỘ: TRUNG CẤP

Ban hành kèm theo Quyết định số: /QĐ-... ngày……tháng….năm ..... …… ..... của……

LƯU HÀNH NỘI BỘ

Đà Lạt, năm 2017

TUYÊN BỐ BẢN QUYỀN

LỜI GIỚI THIỆU

Để đáp ứng nhu cầu học tập của người học, chúng tôi đã tiến hành biên soạn giáo trình cho môn học Cấu trúc dữ liệu và giải thuật. Tài liệu này được biên soạn theo đề cương chi tiết môn học Cấu trúc dữ liệu và giải thuật theo chương trình đào tạo nghề Công nghệ thông tin (Ứng dụng phần mềm) trình độ cao đẳng.

Mục tiêu của giáo trình nhằm giúp các bạn sinh viên chuyên ngành có một tài liệu cô động dùng làm tài liệu học tập, nhưng chúng tôi cũng không loại trừ toàn bộ các đối tượng khác tham khảo. Chúng tôi hi vọng người đọc sẽ tìm thấy được những kiến thức bổ ích trong giáo trình này.

Trong phạm vi giao trình này, chúng tôi giới thiệu các kiến thức, kỹ năng phân tích được các loại dữ liệu, giải thuật và kết hợp được dữ liệu và giải thuật. Kiến thức và kỹ năng cài đặt các thuật toán sắp xếp và tìm kiếm. Kiến thức và kỹ năng cài đặt các thuật toán, các cấu trúc dữ liệu: mảng, danh sách, danh sách liên kết.

Trong quá trình biên soạn, mặc dù đã cố gắng tham khảo nhiều tài liệu và giáo trình khác nhưng tác giả không khỏi tránh được những thiếu sót và hạn chế. Tác giả chân thành mong đợi những nhận xét, đánh giá và góp ý để cuốn giáo trình ngày một hoàn thiện hơn.

Tài liệu này được thiết kế theo từng mô đun/ môn học thuộc hệ thống mô đun/môn học để đào tạo hoàn chỉnh nghề Công nghệ thông tin (Ứng dụng phần mềm) trình độ cao đẳng và được dùng làm Giáo trình cho học viên trong các khoá đào tạo, cũng có thể được sử dụng cho đào tạo ngắn hạn hoặc cho các công nhân kỹ thuật, các nhà quản lý và người sử dụng nhân lực tham khảo.

Đà Lạt, ngày 07 tháng 7 năm 2017

Tham gia biên soạn

1. Chủ biên Ngô Thiên Hoàng
2. Phạm Đình Nam
3. Trương Thị Thanh Thảo
4. Nguyễn Quỳnh Nguyên
5. Phan Ngọc Bảo

MỤC LỤC

Trang

GIÁO TRÌNH 1
LỜI GIỚI THIỆU 2
CHƯƠNG 1 THIẾT KẾ VÀ PHÂN TÍCH GIẢI THUẬT 5
1.1. Mở đầu 5
1.2 Thiết kế giải thuật 6
1.3. Phân tích gải thuật 9
1.4 Thiết kế và phân tích giải thuật và một số ví dụ 11
1.4. Bài tập 26
CHƯƠNG 2: CÁC KIỂU DỮ LIỆU CƠ SỞ 29
2.1. Các kiểu dữ liệu cơ bản 29
2.2 Kiểu dữ liệu có cấu trúc 43
2.3. Kiểu hợp 51
2.4 Kiểu tập tin (file) 55
2.5. Các kiểu dữ liệu khác 67
2.6. Bài tập 70
CHƯƠNG 3: MẢNG, DANH SÁCH VÀ CÁC KIỂU DỮ LIỆU TRỪU TƯỢNG 71
3.1 Mảng 71
3.2. Danh sách liên kết 80
2.3. Các kiểu dữ liệu trừu tượng 149
Bài tập 157
CHƯƠNG 4: SẮP XẾP 159
2.1. Sắp xếp kiểu chọn, chèn, nổi bọt 159
2.2. Sắp xếp bằng phương pháp trộn 170
2.3. Sắp xếp nhanh (Quick sort) 174
Bài Tập: 176
CHƯƠNG 5: TÌM KIẾM 178
5.1 Tìm kiếm tuần tự 178
5.2 Tìm kiếm nhị phân 179
Bài tập 181
TÀI LIỆU THAM KHẢO 182

GIÁO TRÌNH MÔ ĐUN/MÔN HỌC

Tên môn học: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Mã môn học: MH28

I. Vị trí, tính chất của môn học:

1. Vị trí: Môn học được bố trí sau khi sinh viên học xong các môn học/môn học Lập trình căn bản.

2. Tính chất: Cấu trúc dữ liệu và giải thuật là môn học tự chọn, môn kỹ thuật cơ sở áp dụng cho trình độ Cao đẳng Công nghệ thông tin (ứng dụng phần mềm).

II. Mục tiêu môn học:

1. Về kiến thức:

- Xác định được mối quan hệ giữa cấu trúc dữ liệu và giải thuật trong việc xây dựng chương trình;

- Trình bày được ý nghĩa, cấu trúc, cách khai báo, các thao tác của các loại cấu trúc dữ liệu: mảng, danh sách liên kết, các giải thuật cơ bản xử lý các cấu trúc dữ liệu đó;

2. Về kỹ năng:

- Xây dựng được cấu trúc dữ liệu và mô tả tường minh các giải thuật cho một số bài toán ứng dụng cụ thể;

- Cài đặt được một số giải thuật trên ngôn ngữ lập trình C/ C++;

3. Về năng lực tự chủ và trách nhiệm:

- Có khả năng tự nghiên cứu, tự học, tham khảo tài liệu liên quan đến môn học để vận dụng vào hoạt động hoc tập.

- Vận dụng được các kiến thức tự nghiên cứu, học tập và kiến thức, kỹ năng đã được học để hoàn thiện các kỹ năng liên quan đến môn học một cách khoa học, đúng quy định

III. Nội dung môn học:

CHƯƠNG 1 THIẾT KẾ VÀ PHÂN TÍCH GIẢI THUẬT

Mã bài: MH 28.1

Mục tiêu:

- Xác định được mối quan hệ giữa cấu trúc dữ liệu và giải thuật;
- Xác định được tiến trình phân tích và thiết kế thuật toán;
- Trình bày được cách đánh giá độ phức tạp thuật toán;
- Xây dựng được một số giải thuật cơ bản;
- Viết tường minh một số giải thuật;
- Nghiêm túc, tỉ mỉ trong việc học và vận dụng vào làm bài tập.

1.1. Mở đầu

Hằng ngày chúng ta xử lý công việc theo một kế hoạch đã định sẵn bao gồm các bước để thực hiện. Chẳng hạn, để in một văn bản viết tay chúng ta phải thực hiện hai bước quan trọng: Gõ văn bản và In văn bản ra giấy, hai bước này không thể thay đổi thứ tự được. Đây chính là ý tưởng về một giải thuật. Như vậy, *giải thuật là một hệ thống chặt chẽ và rõ ràng các quy tắc nhằm xác định một dãy các thao tác (các bước) trên những đối tượng sao cho một số hữu hạn bước thực hiện các thao tác, chúng ta sẽ đạt được mục tiêu định trước.* Khi thiết kế một giải thuật, chúng ta phải đảm bảo đạt được các đặc trưng của một giải thuật:

a) Tính xác định: Ở mỗi bước của giải thuật, các thao tác phải hết sức rõ ràng. Không thể gây sự lẫn lộn, nhập nhằng, tuỳ tiện.

b) Tính hữu hạn dừng: Một giải thuật bao giờ cũng phải dừng lại sau một số hữu hạn các bước. Có thể số lượng các bước là rất lớn.

c) Tính đúng đắn: Sau khi thực hiện và thuật toán dừng lại, chúng ta phải đạt được kết quả yêu cầu.

d) Tính phổ biến: Thuật toán có thể giải bất kỳ bài toán nào trong cùng một lớp các bài toán, có nghĩa là thuật toán có thể xử lý với các dữ liệu khác nhau.

e) Tính có đại lượng vào và ra: Khi bắt đầu thuật toán lúc nào cũng phải xác định được đại lượng vào, thường được lấy từ tập xác định cho trước; Sau khi kết thúc, thuật toán phải cho ra một hay một số đại lượng ra tuỳ theo chức năng mà thuật toán đảm nhiệm.

f) Tính hiệu quả: Được đánh giá trên cá đại lượng: Bộ nhớ cần có, số các phép tính cần thực hiện, thời gian cần thiết để chạy, có dễ hiểu đối với con người không, dễ cài đặt hay không.

Trong đó 8 đặc trưng đầu phải đáp ứng được.

Tuy nhiên, giải thuật sẽ tác động trên dữ liệu nào để đưa đến kết quả mong muốn. Việc "tổ chức" các phần tử dữ liệu theo mối quan hệ của chúng theo cấu trúc thích hợp để thực hiện xử lý thuận lợi hơn, hiệu quả cao hơn cũng cực kỳ quan trọng. Do đó, *kết hợp các kiểu dữ liệu đơn giản để tạo một kiểu dữ liệu mới phù hợp hơn, kiểu dữ liệu mới này được gọi là một cấu trúc dữ liệu.*

1.2 Thiết kế giải thuật

Thực hiện một đề án tin học là chuyển bài toán thực tế thành bài toán có thể giải quyết trên máy tính. Một bài toán thực tế bất kỳ đều bao gồm các đối tượng dữ liệu và các yêu cầu xử lý trên những đối tượng đó. Vì thế, để xây dựng một mô hình tin học phản ánh được bài toán thực tế cần chú trọng đến hai vấn đề :

Một bài toán cụ thể khi được chuyển để giải quyết trên máy tính đều bao gồm các đốit tượng dữ liệu và các yêu cầu xử lý trên các đối tượng đó. Do đó, chúng ta cần phải chú trọng:

- Tổ chức biểu diễn các đối tượng thực tế :

Các thành phần dữ liệu thực tế đa dạng, phong phú và thường chứa đựng những quan hệ nào đó với nhau, do đó trong mô hình tin học của bài toán, cần phải tổ chức, xây dựng các cấu trúc thích hợp nhất sao cho vừa có thể phản ánh chính xác các dữ liệu thực tế này, vừa có thể dễ dàng dùng máy tính để xử lý. Công việc này được gọi là xây dựng *cấu trúc dữ liệu* cho bài toán.

- Xây dựng các thao tác xử lý dữ liệu:

Từ những yêu cầu xử lý thực tế, cần tìm ra các giải thuật tương ứng để xác định trình tự các thao tác máy tính phải thi hành để cho ra kết quả mong muốn, đây là bước xây dựng *giải thuật* cho bài toán.

Tuy nhiên, khi giải quyết một bài toán trên máy tính, chúng ta thường có khuynh hướng chỉ chú trọng đến việc xây dựng giải thuật mà quên đi tầm quan trọng của việc tổ chức dữ liệu trong bài toán. Giải thuật phản ánh các phép xử lý , còn đối tượng xử lý của giải thuật lại là dữ liệu, chính dữ liệu chứa đựng các thông tin cần thiết để thực hiện giải thuật. Để xác định được giải thuật phù hợp cần phải biết nó tác động đến loại dữ liệu nào (ví dụ để làm nhuyễn các hạt đậu , người chúng ta dùng cách xay chứ

không băm bằng dao, vì đậu sẽ văng ra ngoài) và khi chọn lựa cấu trúc dữ liệu cũng cần phải hiểu rõ những thao tác nào sẽ tác động đến nó (ví dụ để biểu diễn các Doanh thu của cửa hàng người chúng ta dùng số thực thay vì chuỗi ký tự vì còn phải thực hiện thao tác tính trung bình từ những Doanh thu đó). Như vậy trong một đề án tin học, giải thuật và cấu trúc dữ liệu có mối quan hệ chặt chẽ với nhau, được thể hiện qua công thức :

Cấu trúc dữ liệu + Giải thuật = Chương trình

Khi chọn lựu một cấu trúc dữ liệu, sẽ có những giải thuật hay phép toán tương ứng, phù hợp. Khi cấu trúc dữ liệu thay đổi thường giải thuật cũng phải thay đổi theo để tránh việc xử lý gượng ép, thiếu tự nhiên trên một cấu trúc không phù hợp. Hơn nữa, một cấu trúc dữ liệu tốt sẽ giúp giải thuật xử lý trên đó có thể phát huy tác dụng tốt hơn, vừa đáp ứng nhanh vừa tiết kiệm vật tư, giải thuật cũng dễ hiểu và đơn giản hơn.

Như vậy, giải thuật và cấu trúc dữ liệu có liên hệ mật thiết với nhau; Nếu chúng ta thay đổi cấu trúc dữ liệu thì giải thuật cũng sẽ thay đổi theo để xử lý phù hợp với cấu trúc dữ liệu mới nhằm đưa đến kết quả mong muốn.

Ví dụ 1: Một chương trình quản lý doanh thu của các cửa hàng cần lưu trữ các doanh thu của 3 cửa hàng. Do mỗi cửa hàng có 4 doanh thu ứng với 4 quý khác nhau nên dữ liệu có dạng bảng như sau:

Cửa hàng	Quý 1	Quý 2	Quý 3	Quý 4
Số 1	7	10	8	2
Số 2	8	0	10	4
Số 3	6	3	7	4

Chỉ xét thao tác xử lý là xuất doanh thu các Quý của từng cửa hàng.

Giả sử có các phương án tổ chức lưu trữ sau:

Phương án 1 : Sử dụng mảng một chiều

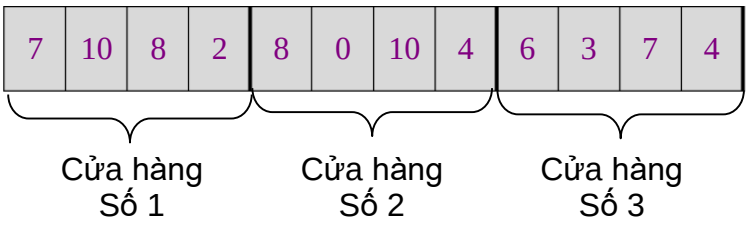
Có tất cả 3(cửa hàng)*4(Quý) = 12 doanh thu cần lưu trữ, do đó khai báo mảng result như sau :

int result [12] = {7, 10, 8, 2,

8, 0, 10, 4,

6, 3, 7, 4};

khi đó trong mảng result các phần tử sẽ được lưu trữ như sau:



Và truy xuất Doanh thu Quý j của cửa hàng i - là phần tử tại (dòng i, cột j) trong bảng - phải sử dụng một công thức xác định chỉ số tương ứng trong mảng result:

bảng doanh thu(dòng i, cột j) = result[((i-1)*số cột) + j]

Ngược lại, với một phần tử bất kỳ trong mảng, muốn biết đó là Doanh thu của cửa hàng nào, Quý gì, phải dùng công thức xác định sau

result[i] = bảng doanh thu (dòng((i / số cột) +1), cột (i % số cột))

Với phương án này, thao tác xử lý được cài đặt như sau :

void XuatDoanhThu() //Xuất Doanh thu của tất cả cửa hàng

```
{
    const int so_quy = 4;
    int cuahang,quy;
    for (int i=0; i<12; i+)
    {
    cuahang = i/so_cuahang;
quy = i % so_quy+1;
printf("Doanh thu Quý %d của cua hang %d là: %d", cuahang, sv,result[i]);
    }
    }
```

Phương án 2 : Sử dụng mảng 2 chiều

Khai báo mảng 2 chiều result có kích thước 3 dòng* 4 cột như sau :

int result[3][4] ={{ 7, 10, 8, 2},

{ 8, 0, 10, 4},

{ 6, 3, 7, 4 }};

khi đó trong mảng result các phần tử sẽ được lưu trữ như sau :

	Cột 0	Cột 1	Cột 2	Cột 3
Dòng 0	result[0][0] =7	result[0][1] =10	result[0][2] =8	result[0][3] =2
Dòng 1	result[1][0] =8	result[1][1] =0	result[1][2] =10	result[1][3] =4
Dòng 2	result[2][0] =6	result[2][1] =3	result[2][2] =7	result[2][3] =4

Và truy xuất Doanh thu Quý j của cửa hàng i - là phần tử tại (dòng i, cột j) trong bảng - cũng chính là phần tử nằm ở vị trí (dòng i, cột j) trong mảng

bảngDoanh thu(dòng i,cột j) = result[i] [j]

Với phương án này, thao tác xử lý được cài đặt như sau :

```
void XuatDoanhThu() //Xuất Doanh thu của tất cả cửa hàng
{
int so_quy = 4, so_cuahang =3;
for ( int i=0; i<so_cuahang; i+)
for ( int j=0; j<so_quy; j++)
   printf("Doanh thu Quý %d của cửa hàng %d là: %d", j, i,result[i][j]);
}
```

Chúng ta có thể thấy rõ phương án 2 cung cấp một cấu trúc lưu trữ phù hợp với dữ liệu thực tế hơn phương án 1, và do vậy giải thuật xử lý trên cấu trúc dữ liệu của phương án 2 cũng đơn giản, tự nhiên hơn.

1.3. Phân tích gải thuật

Thuật ngữ kiểu dữ liệu (hay kiểu - data type), cấu trúc dữ liệu (data structure), kiểu dữ liệu trừu tượng (abstract data type) nghe như nhau, nhưng chúng có ý nghĩa rất khác nhau. Trong một ngôn ngữ lập trình, kiểu dữ liệu của một biến là tập hợp các giá trị mà biến đó có thể nhận. Như vậy, định nghĩa kiểu dữ liệu như sau:

Kiểu dữ liệu T được xác định bởi một bộ <V,O> , với :

V : tập các giá trị hợp lệ mà một đối tượng kiểu T có thể lưu trữ

O : tập các thao tác xử lý có thể thi hành trên đối tượng kiểu T.

Ví du 2: Giả sử có kiểu dữ liệu mẫu tự = <Vc ,Oc> với

Vc = { a-z,A-Z}

Oc = { lấy mã ASCII của ký tự, biến đổi ký tự thường thành ký tự hoa…}

Giả sử có kiểu dữ liệu số nguyên = <Vi ,Oi> với

Vi = { -32768..32767}

Oi = { +, -, *, /, %}

Như vậy, muốn sử dụng một kiểu dữ liệu cần nắm vững cả nội dung dữ liệu được phép lưu trữ và các xử lý tác động trên đó.

Các thuộc tính của 1 kiểu dữ liệu bao gồm:

- Tên kiểu dữ liệu
- Miền giá trị
- Kích thước lưu trữ
- Tập các toán tử tác động lên kiểu dữ liệu.

Một kiểu dữ liệu trừu tượng là một mô hình toán học cùng với một tập hợp các phép toán trên nó. Có thể nói kiểu dữ liệu trừu tượng là một kiểu dữ liệu do chúng ta định nghĩa ở mức khái niệm (conceptual), nó chưa được cài đặt cụ thể bằng một ngôn ngữ lập trình. Như đã dẫn ra ở trên, chúng ta dùng kiểu dữ liệu trừu tượng để thiết kế giải thuật, nhưng để cài đặt giải thuật vào một ngôn ngữ lập trình chúng ta phải tìm cách biểu diễn kiểu dữ liệu trừu tượng trên các kiểu dữ liệu và toán tử do ngôn ngữ lập trình cung cấp. Như vậy khi biểu diễn (cài đặt) một kiểu dữ liệu trừu tượng trong một ngôn ngữ lập trình, chúng ta dùng các cấu trúc dữ liệu, đó là một tập hợp các biến có thể thuộc một vài kiểu khác nhau được nối kết với nhau.

Ô (cell) là khối cơ bản để xây dựng các cấu trúc dữ liệu. Một ô có thể xem như là một cái hộp có thể lưu giữ một giá trị của một kiểu dữ liệu cơ sở hoặc là một kiểu dữ liệu hợp thành. Các cấu trúc dữ liệu được tạo ra bằng cách cho tên và cung cách "nối kết" các ô.

Ví dụ 3: mảng (ARRAY) hoặc mẩu tin (RECORD) là các cấu trúc dữ liệu thường gặp trong các ngôn ngữ lập trình. Trong chương sau chúng ta sẽ gặp các cấu trúc có dùng con trỏ (pointer).

Một kiểu dữ liệu trừu tượng (ADT) là một mô hình toán học cùng với một tập hợp các phép toán (operator) được định nghĩa trên mô hình đó. Ví dụ tập hợp số nguyên cùng với các phép toán hợp, giao, hiệu là một kiểu dữ liệu trừu tượng.

Trong một ADT các phép toán có thể thực hiện trên các đối tượng (toán hạng) không chỉ thuộc ADT đó, cũng như kết quả không nhất thiết phải thuộc ADT. ADT là sự tổng quát hoá của các kiểu dữ liệu nguyên thuỷ, giống như hàm là sự tổng quát hoá của các phép toán nguyên thuỷ.

1.4 Thiết kế và phân tích giải thuật và một số ví dụ

Trước khi giải quyết một bài toán, chúng ta cần phải phân tích và thiết kế giải thuật hợp lý. Một bài toán có thể có rất nhiều giải thuật được thiết kế để giải quyết. Tuy nhiên, để thiết kế một giải thuật giải quyết hiệu quả, thường người chúng ta sẽ tuân theo ba bước sau:

- Bước 1: Người thiết kế phải xác định được dữ liệu đầu vào của giải thuật và lựa chọn phương pháp phân tích thích hợp. Chúng ta thường không thu được một hàm chính xác giải thuật dựa trên đầu vào. Trong thực tế chúng ta luôn cố gắng chứng minh thời gian chạy của giải thuật luôn nhỏ hơn một "chặn trên" bất chấp dữ liệu đầu vào như thế nào.

- Bước 2: Người thiết kế phải nhận ra các thao tác trừu tượng của giải thuật để tách biệt sự phân tích với sự cài đặt. Ví dụ, chúng ta tách biệt sự nghiên cứu có bao nhiêu phép tính thực hiện trong một giải thuật ra khỏi sự khác định cần bao nhiêu thời gian để máy tính giải quyết; yếu tố thứ nhất được xác định bởi tính chất của giải thuật, yếu tố thứ hai lại được xác định bởi tính chất của máy tính. Sự tách biệt này cho phép chúng ta so sánh các giải thuật một cách độc lập với sự cài đặt cụ thể hay độc lập với một máy tính cụ thể.

- Bước 3: Người thiết kế cần phải có sự phân tích về mặt toán học, với mục đích tìm ra các giá trị trung bình và trường hợp xấu nhất cho mỗi đại lượng cơ bản. Chúng ta không gặp khó khăn khi tìm một chặn trên cho thời gian chạy chương trình, vấn đề ở chỗ là phải tìm ra chặn trên tốt nhất.

Trong khi giải một bài toán chúng ta có thể có một số giải thuật khác nhau, vấn đề là cần phải đánh giá các giải thuật đó để lựa chọn một giải thuật tốt (nhất). Thông thường thì chúng ta sẽ căn cứ vào các tiêu chuẩn sau:

- Giải thuật đúng đắn. (1)
- Giải thuật đơn giản. (2)

- Giải thuật thực hiện nhanh. (3)

Với yêu cầu (1), để kiểm tra tính đúng đắn của giải thuật chúng ta có thể cài đặt giải thuật đó và cho thực hiện trên máy với một số bộ dữ liệu mẫu rồi lấy kết quả thu được so sánh với kết quả đã biết. Thực ra thì cách làm này không chắc chắn bởi vì có thể giải thuật đúng với tất cả các bộ dữ liệu chúng ta đã thử nhưng lại sai với một bộ dữ liệu nào đó. Vả lại cách làm này chỉ phát hiện ra giải thuật sai chứ chưa chứng minh được là nó đúng. Tính đúng đắn của giải thuật cần phải được chứng minh bằng toán học. Tất nhiên điều này không đơn giản và do vậy chúng ta sẽ không đề cập đến ở đây.

Khi chúng ta viết một chương trình để sử dụng một vài lần thì yêu cầu (2) là quan trọng nhất. Chúng ta cần một giải thuật dễ viết chương trình để nhanh chóng có được kết qủa , thời gian thực hiện chương trình không được đề cao vì dù sao thì chương trình đó cũng chỉ sử dụng một vài lần mà thôi.

Tuy nhiên khi một chương trình được sử dụng nhiều lần thì thì yêu cầu tiết kiệm thời gian thực hiện chương trình lại rất quan trọng đặc biệt đối với những chương trình mà khi thực hiện cần dữ liệu nhập lớn do đó yêu cầu (3) sẽ được xem xét một cách kỹ càng. Chúng ta gọi nó là hiệu quả thời gian thực hiện của giải thuật.

Một phương pháp để xác định hiệu quả thời gian thực hiện của một giải thuật là lập trình nó và đo lường thời gian thực hiện của hoạt động trên một máy tính xác định đối với tập hợp được chọn lọc các dữ liệu vào.

Thời gian thực hiện không chỉ phụ thuộc vào giải thuật mà còn phụ thuộc váo tập các chỉ thị của máy tính, chất lượng của máy tính và kỹ xảo của người lập trình. Sự thi hành cũng có thể điều chỉnh để thực hiện tốt trên tập đặc biệt các dữ liệu vào được chọn. Để vượt qua các trở ngại này, các nhà khoa học máy tính đã chấp nhận tính phức tạp của thời gian được tiếp cận như một sự đo lường cơ bản sự thực thi của giải thuật. Thuật ngữ tính hiệu quả sẽ đề cập đến sự đo lường này và đặc biệt đối với sự phức tạp thời gian trong trường hợp xấu nhất.

Thời gian thực hiện một chương trình được xác định dựa trên một hàm của kích thước dữ liệu vào, ký hiệu T(n) trong đó n là kích thước (độ lớn) của dữ liệu vào.

Ví dụ 1: Chương trình tính tổng của n số có thời gian thực hiện là T(n) = cn trong đó c là một hằng số.

Thời gian thực hiện chương trình là một hàm không âm, tức là $T(n) \geq 0 \ \forall n \geq 0$.

Đơn vị của T(n) không phải là đơn vị đo thời gian bình thường như giờ, phút giây... mà thường được xác định bởi số các lệnh được thực hiện trong một máy tính lý tưởng.

Ví dụ 2: Khi chúng ta nói thời gian thực hiện của một chương trình là T(n) = cn thì có nghĩa là chương trình ấy cần cn chỉ thị thực thi.

Nhìn chung, thời gian thực hiện chương trình không chỉ phụ thuộc vào kích thước mà còn phụ thuộc vào tính chất của dữ liệu vào. Nghĩa là dữ liệu vào có cùng kích thước nhưng thời gian thực hiện chương trình có thể khác nhau. Chẳng hạn chương trình sắp xếp dãy số nguyên tăng dần, khi chúng ta cho vào dãy có thứ tự thì thời gian thực hiện khác với khi chúng ta cho vào dãy chưa có thứ tự, hoặc khi chúng ta cho vào một dãy đã có thứ tự tăng thì thời gian thực hiện cũng khác so với khi chúng ta cho vào một dãy đã có thứ tự giảm.

Vì vậy thường chúng ta coi T(n) là thời gian thực hiện chương trình trong trường hợp xấu nhất trên dữ liệu vào có kích thuớc n, tức là: T(n) là thời gian lớn nhất để thực hiện chương trình đối với mọi dữ liệu vào có cùng kích thước n.

Chúng ta nói rằng hàm không âm T(n) có tỷ suất tăng (growth rate) f(n) nếu tồn tại các hằng số c và n0 sao cho $T(n) \leq f(n)$ với mọi $n \geq n0$.

Chúng ta có thể chứng minh được rằng "Cho một hàm không âm T(n) bất kỳ, chúng ta luôn tìm được tỷ suất tăng f(n) của nó".

Ví dụ 3: Giả sử T(0) = 1, T(1) = 4 và tổng quát T(n) = (n+1)2. Đặt n0 = 1 và c = 4 thì với mọi $n \geq 1$ chúng ta dễ dàng chứng minh rằng $T(n) = (n+1)2 \leq 4n2$ với mọi $n \geq 1$, tức là tỷ suất tăng của T(n) là n2.

Ví dụ 4: Tỷ suất tăng của hàm T(n) = 4n4 + 2n2 là n4. Thực vậy, cho n0 = 0 và c = 6 chúng ta dễ dàng chứng minh rằng với mọi $n \geq 0$ thì $4n4 + 2n2 \leq 6n3$

Như vậy, độ phức tạp của thuật toán là gì?

Trước tiên, chúng ta xét trong trường hợp có hai giải thuật P1 và P2 với thời gian thực hiện tương ứng là T1(n) = 100n2 (với tỷ suất tăng là n2) và T2(n) = 5n3 (với tỷ suất tăng là n3). Giải thuật nào sẽ thực hiện nhanh hơn? Câu trả lời phụ thuộc vào kích thước dữ liệu vào. Với $n < 20$ thì P2 sẽ nhanh hơn P1 (T2<T1), do hệ số của 5n3 nhỏ hơn hệ số của 100n2 (5<100). Nhưng khi $n > 20$ thì ngược lại do số mũ của 100n2 nhỏ hơn số mũ của 5n3 (2<3). Ở đây chúng ta chỉ nên quan tâm đến trường

hợp n>20 vì khi n<20 thì thời gian thực hiện của cả P1 và P2 đều không có sự khác biệt lớn giữa T1 và T2. Một cách hợp lý là chúng ta xét tỷ suất tăng của hàm thời gian thực hiện chương trình thay vì xét chính bản thân thời gian thực hiện.

Cho một hàm T(n), T(n) gọi là có độ phức tạp f(n) nếu tồn tại các hằng c, N0 sao cho $T(n) \leq cf(n)$ với mọi $n \geq N0$ (tức là T(n) có tỷ suất tăng là f(n)) và kí hiệu T(n) là O(f(n)) (đọc là "ô của f(n)")

Ví dụ 5: T(n)= (n+1)2 có tỷ suất tăng là n2 nên T(n)= (n+1)2 là O(n2).

Chú ý: O(c.f(n))=O(f(n)) với c là hằng số. Đặc biệt O(c)=O(1).

Nói cách khác độ phức tạp tính toán của giải thuật là một hàm chặn trên của hàm thời gian. Vì hằng nhân tử c trong hàm chặn trên không có ý nghĩa nên chúng ta có thể bỏ qua vì vậy hàm thể hiện độ phức tạp có các dạng thường gặp sau: log2n, n, nlog2n, n2, n3, 22, n!, nn. Ba hàm cuối cùng chúng ta gọi là dạng hàm mũ, các hàm khác gọi là hàm đa thức. Một giải thuật mà thời gian thực hiện có độ phức tạp là một hàm đa thức thì chấp nhận được tức là có thể cài đặt để thực hiện, còn các giải thuật có độ phức tạp hàm mũ thì phải tìm cách cải tiến giải thuật.
Khi nói đến độ phức tạp của giải thuật là chúng ta muốn nói đến hiệu quả của thời gian thực hiện của chương trình nên chúng ta có thể xem việc xác định thời gian thực hiên của chương trình chính là xác định độ phức tạp của giải thuật.

Để tính được độ phức tạp của thuật toán, chúng ta xử lý theo các quy tắc và phương pháp sau.

-Qui tắc cộng:

Nếu T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1 và P2; và T1(n)=O(f(n)), T2(n)=O(g(n) thì thời gian thực hiện của đoạn hai chương trình đó nối tiếp nhau là T(n)=O(max(f(n),g(n)))

Đoạn lệnh	Độ phức tạp
P1	O(f(n))
P2	O(g(n))
P1; P2	O(max(f(n),g(n))

Ví dụ 6: Lệnh gán x=15 tốn một hằng thời gian hay O(1)

Lệnh đọc dữ liệu scanf(&x) tốn một hằng thời gian hay O(1)

Vậy thời gian thực hiện đoạn lệnh:

```
x=15;
scanf(&x);
```

là O(max(1,1))=O(1).

- Qui tắc nhân:

Nếu T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1 và P2 và T1(n) = O(f(n)), T2(n) = O(g(n) thì thời gian thực hiện của đoạn hai đoạn chương trình đó lồng nhau là T(n) = O(f(n).g(n))

Đoạn lệnh	Độ phức tạp
P1	O(f(n))
P2	O(g(n))

P1
{
P2;
}
	O(f(n)*g(n))

- Qui tắc tổng quát để phân tích một chương trình:

+ Thời gian thực hiện của mỗi lệnh gán, cin, cout, printf, scanf là O(1)

+ Thời gian thực hiện của một chuỗi tuần tự các lệnh được xác định bằng qui tắc cộng. Như vậy thời gian này là thời gian thi hành một lệnh nào đó lâu nhất trong chuỗi lệnh.

+ Thời gian thực hiện cấu trúc if là thời gian lớn nhất thực hiện lệnh sau if hoặc sau else và thời gian kiểm tra điều kiện. Thường thời gian kiểm tra điều kiện là O(1).

+ Thời gian thực hiện vòng lặp là tổng (trên tất cả các lần lặp) thời gian thực hiện thân vòng lặp. Nếu thời gian thực hiện thân vòng lặp không đổi thì thời gian thực hiện vòng lặp là tích của số lần lặp với thời gian thực hiện thân vòng lặp.

Ví dụ 7: Tính thời gian thực hiện của đoạn chương trình

```
void Bubble (int a[], int n);
int i,j,temp;
{
for (i=0;i<=n-2;i++) //(1)
for (j=n-1;j>=i+1;j--) //(2)
if (a[j-1]>a[j])   //(3)
{
// đổi chỗ a[i], a[j]
temp=a[j-1]; //(4)
a[j-1]=a[j]; //(5)
a[j]=temp; } //(6)
}
```

Cả ba lệnh đổi chỗ {4} {5} {6} tốn O(1) thời gian, do đó lệnh {3} tốn O(1).

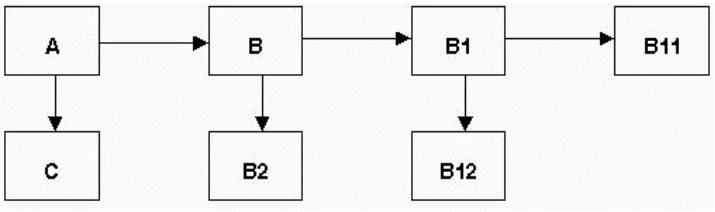
Vòng lặp {2} thực hiện (n-i) lần, mỗi lần O(1) do đó vòng lặp {2} tốn O((n-i).1)=O(n-i).

Vòng lặp {1} lặp (n-1) lần vậy độ phức tạp của giải thuật là:

$$T(n)=\sum_{i=1}^{n-1}(n-i)=\frac{n(n-1)}{2}=O(n^2)$$

Độ phức tạp của chương trình có gọi chương trình con không đệ qui Nếu chúng ta có một chương trình với các chương trình con không đệ quy, để tính thời gian thực hiện của chương trình, trước hết chúng ta tính thời gian thực hiện của các chương trình con không gọi các chương trình con khác. Sau đó chúng ta tính thời gian thực hiện của các chương trình con chỉ gọi các chương trình con mà thời gian thực hiện của chúng đã được tính. Chúng ta tiếp tục quá trình đánh giá thời gian thực hiện của mỗi chương trình con sau khi thời gian thực hiện của tất cả các chương trình con mà nó gọi đã được đánh giá. Cuối cùng chúng ta tính thời gian cho chương trình chính.

Giả sử chúng ta có một hệ thống các chương trình gọi theo sơ đồ sau:



Chương trình A gọi hai chương trình con là B và C, chương trình B gọi hai chương trình con là B1 và B2, chương trình B1 gọi hai chương trình con là B11 và B12.

Để tính thời gian thực hiện của A, chúng ta tính theo các bước sau:

- Tính thời gian thực hiện của C, B2, B11 và B12.
- Tính thời gian thực hiện của B1.
- Tính thời gian thực hiện của B.
- Tính thời gian thực hiện của A.

Ví dụ 4: Chúng ta có thể viết lại chương trình sắp xếp bubble sort như sau:

```
void swap (int &x, int  &y)
{
 int temp;
 temp = x;
 x = y;
 y = temp;
}
voidn Bubble (int a[], int n)
{ int i,j;
{
 {1} for (i=0;i<=n-2;i++)
 {2} for (j=n-1;j>=i+1;j--)
 {3} if a[j-1]>a[j] then swap(a[j-1], a[j]);
}
```

Trong cách viết trên, chương trình Bubble gọi chương trình con swap, do đó để tính thời gian thực hiện của Bubble, trước hết chúng ta cần tính thời gian thực hiện của Swap. Dễ thấy thời gian thực hiện của Swap là O(1) vì nó chỉ bao gồm 3 lệnh gán.

Trong Bubble, lệnh {3} gọi swap nên chỉ tốn O(1), lệnh {2} thực hiện n-i lần, mỗi lần tốn O(1) nên tốn O(n-i). Lệnh {1} thực hiện n-1 lần nên

$$T(n)=\sum_{i=1}^{n-1}(n-i)=\frac{n(n-1)}{2}=O(n^2)$$

Trong trường hợp chương trình có đệ quy, chúng ta thực hiện theo phương pháp sau:

Trước hết, chúng ta cần thành lập các phương trình đệ quy, sau đó giải phương trình đệ quy, nghiệm của phương trình đệ quy sẽ là thời gian thực hiện của chương trình đệ quy.

Phương trình đệ quy là một phương trình biểu diễn mối liên hệ giữa T(n) và T(k), trong đó T(n) là thời gian thực hiện chương trình với kích thước dữ liệu nhập là n, T(k) thời gian thực hiện chương trình với kích thước dữ liệu nhập là k, với k < n. Để thành lập được phương trình đệ quy, chúng ta phải căn cứ vào chương trình đệ quy.

Ví dụ 8: Xét hàm tính giai thừa viết bằng giải thuật đệ quy như sau:

```
int giaithua(int n)
{
 if (n=0)
   return 1;
 else
  return  n* giaithua(n-1);
}
```

Gọi T(n) là thời gian thực hiện việc tính n giai thừa, thì T(n-1) là thời gian thực hiện việc tính n-1 giai thừa. Trong trường hợp n=0 thì chương trình chỉ thực hiện một lệnh gán Giai_thua:=1, nên tốn O(1), do đó chúng ta có T(0) = C1. Trong trường hợp n>0 chương trình phải gọi đệ quy giaithua(n-1), việc gọi đệ quy này tốn T(n-1), sau khi có kết quả của việc gọi đệ quy, chương trình phải nhân kết quả đó với n và gán cho giaithua. Thời gian để thực hiện phép nhân và phép gán là một hằng C2. Vậy chúng ta có

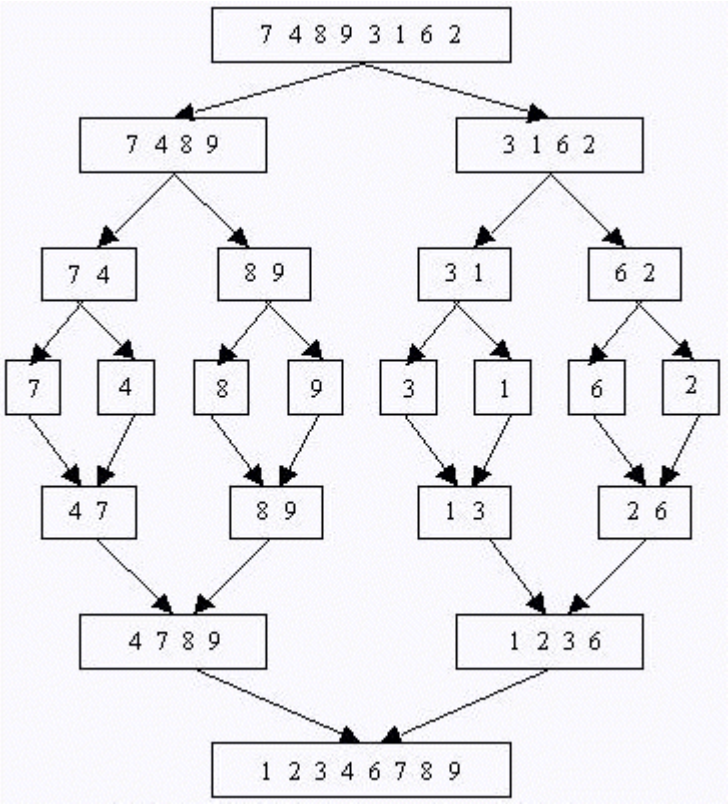
$$T(n)=\begin{cases}C1 & \text{nếu n=0}\\ T(n-1)+C2 & \text{nếu n>0}\end{cases}$$

Đây là phương trình đệ quy để tính thời gian thực hiện của chương trình đệ quy giaithua.

Ví dụ 99: Chúng ta xét thủ tục MergeSort một cách phác thảo như sau:

```
list MergeSort (list L, int n)
{  list L1,L2;
 if (n == 1) then return(L)
 else
 {
 Chia L thành 2 nửa L1 và L2, mỗi một nửa có độ dài n/2;
 return(Merge(MergeSort (L1, n/2), MergeSort(L2, n/2)));
}
}
```

Chẳng hạn để sắp xếp danh sách L gồm 8 phần tử 7, 4, 8, 9, 3, 1, 6, 2 chúng ta có mô hình minh họa của MergeSort như sau:



Hàm MergeSort nhận một danh sách có độ dài n và trả về một danh sách đã được sắp xếp. Thủ tục Merge nhận hai danh sách đã được sắp L1 và L2 mỗi danh sách có độ dài n/2, trộn chúng lại với nhau để được một danh sách gồm n phần tử có thứ tự. Giải thuật chi tiết của Merge chúng ta sẽ bàn sau, chúng ta chỉ để ý rằng thời gian để Merge các danh sách có độ dài n/2 là O(n).

Gọi T(n) là thời gian thực hiện MergeSort một danh sách n phần tử thì T(n/2) là thời gian thực hiện MergeSort một danh sách n/2 phần tử, chúng ta có thể viết phương trình đệ quy như sau:

$$T(n)=\begin{cases} C_1 & \text{nếu n=1} \\ 2T\left(\frac{n}{2}\right)+C_2 n & \text{nếu n>1} \end{cases}$$

Trong đó C1 là thời gian phải tốn khi L có độ dài 1. Trong trường hợp n > 1, thời gian của MergeSort được chia làm hai phần. Phần gọi đệ quy MergeSort một danh sách có độ dài n/2 là T(n/2) do đó chúng ta có 2T(n/2). Phần thứ hai bao gồm phép thử n >1, chia danh sách L thành hai nửa bằng nhau và Merge. Ba thao tác này lấy thời gian không đổi đối với phép thử hoặc tỷ lệ với n đối với ngắt và Merge. Như vậy hằng C2 được chọn và C2n là thời gian tổng để làm các việc đó ngoại trừ gọi đệ quy.

Để giải phương trình đệ quy, chúng ta sử dụng một số phương pháp sau:

- Phương pháp truy hồi
- Phương pháp đoán nghiệm.
- Lời giải tổng quát của một lớp các phương trình đệ quy.

Chúng ta tiến hành nghiên cứu chi tiết các phương pháp này:

- Phương pháp truy hồi :

Dùng đệ quy để thay thế bất kỳ T(m) với m < n vào phía phải của phương trình cho đến khi tất cả T(m) với m > 1 được thay thế bởi biểu thức của các T(1). Vì T(1) luôn là hằng nên chúng ta có công thức của T(n) chứa các số hạng chỉ liên quan đến n và các hằng số.

Giải phương trình.

Ví dụ 9: Giải phương trình:

$$T(n)=\begin{cases} C1 & \text{nếu n=1} \\ 2T\left(\frac{n}{2}\right)+C_2 n & \text{nếu n>1} \end{cases}$$

Chúng ta có:

$$T(n)=2T\left(\frac{n}{2^1}\right)+C_2 n$$

$$T(n) = 2\left[2T\left(\frac{n}{4}\right) + C_2\frac{n}{2}\right] = 4T\left(\frac{n}{4}\right) + 2C_2n$$

$$T(n) = 4\left[2T\left(\frac{n}{8}\right) + C_2\frac{n}{4}\right] + 2C_2n = 8T\left(\frac{n}{8}\right) + 3C_2n$$

…….

$$T(n) = 2^iT\left(\frac{n}{2}\right) + iC_2n$$

Giả sử n = 2k, quá trình suy rộng sẽ kết thúc khi i =k, khi đó chúng ta có:

T(n) = 2kT(1) + kC2n

Vì 2k = n nên k = log2n và với T(1) =C1 nên chúng ta có

T(n) = C1n + C2nlog2n

Hay T(n) là O(nlog2n).

- Phương pháp đoán nghiệm

Chúng ta đoán một nghiệm f(n) và dùng chứng minh quy nạp để chứng tỏ rằng T(n) ≤ f(n) với mọi n.

Thông thường f(n) là một trong các hàm quen thuộc như logn, n, nlog2n, n2, n3, 2n, n!, nn.

Đôi khi chúng ta chỉ đoán dạng của f(n) trong đó có một vài tham số chưa xác định (chẳng hạn f(n) = an2 với a chưa xác định) và trong quá trình chứng minh quy nạp chúng ta sẽ suy diễn ra giá trị thích hợp của các tham số.

Ví dụ 10: Giải phương trình đệ quy

$$T(n) = \begin{cases} C1 \text{ nếu n=1} \\ 2T\left(\frac{n}{2}\right) + C_2n \text{ nếu n>1} \end{cases}$$

Giả sử chúng ta đoán f(n) = anlog2n. Với n = 1 chúng ta thấy rằng cách đoán như vậy không được bởi vì anlog2n có giá trị 0 không phụ thuộc vào giá trị của a. Vì thế chúng ta thử tiếp theo f(n) = anlog2n + b.

Với n = 1 chúng ta có, T(1) = C1 và f(1) = b, muốn T(1) ≤ f(1) thì b ≥C1 (*)

Giả sử rằng T(k) d" aklogk + b với mọi k < n (I.2).Chúng ta sẽ chứng minh T(n) ≤ anlog2n + b

Giả sử n ≥ 2, từ (I.1) chúng ta có T(n) = 2T(n/2) + C2n

Áp dụng (I.2) với k = n/2 < n chúng ta có:

T(n) = 2T(n/2) + C2n ≤ 2[an/2log(n/2) + b] + C2n

T(n) ≤ anlog2n - an + 2b + C2n

T(n) ≤(anlog2n + b) + [b + (C2 - a)n]. Nếu lấy a ≥ C2 + b (**) chúng ta được

T(n) ≤ (anlog2n + b) + [b +(C2 - C2 - b)n]

T(n) ≤ (anlog2n + b) + (1-n) b

T(n) ≤ an log2n + b.

Nếu chúng ta lấy a và b sao cho cả (*) và (**) đều thoả mãn thì T(n) ≤ an log2n + b với mọi n.

Chúng ta giải hệ $\begin{cases} b \geq C_1 \\ a \geq C_2 + b \end{cases}$, để đơn giản chúng ta giải hệ $\begin{cases} b = C_1 \\ a = C_2 + b \end{cases}$

Dễ dàng chúng ta có b = C1 và a= C1 +C2 chúng ta được T(n) ≤ (C1 + C2)nlog2n + C1 với mọi n.

Hay nói cách khác T(n) là O(nlog2n).

- Lời giải tổng quát cho một lớp các phương trình đệ quy

Để giải bài toán kích thước n, chúng ta chia bài toán đã cho thành a bài toán con, mỗi bài tóan con có kích thước n/b. Giải các bài toán con này và tổng hợp kết quả lại để được kết quả của bài toán đã cho. Với các bài toán con chúng ta cũng làm như vậy. Kỹ thuật này sẽ dẫn chúng ta đến một chương trình đệ quy.

Giả thiết rằng mỗi bài toán con kích thước 1 lấy một đơn vị thời gian và thời gian để chia bài toán kích thước n thành các bài toán con kích thước n/b và tổng hợp kết quả từ các bài toán con để được lời giải của bài toán ban đầu là d(n). (Chẳng hạn đối với thí dụ MergeSort, chúng ta có a = b = 2, và d(n) = C2n/C1. Xem C1 là một đơn vị).

Gọi T(n) là thời gian để giải bài toán kích thước n thì chúng ta có phương trình đệ quy:

Gọi T(n) là thời gian để giải bài toán kích thước n thì chúng ta có phương trình đệ quy:

$$\begin{cases} T(1) = 1 \\ T(n) = aT\left(\frac{n}{b}\right) + d(n) \end{cases} \quad \text{(I.1)}$$

Chúng ta sử dụng phương pháp truy hồi để giải phương trình này

$$T(n) = aT(n/b) + d(n)$$

$$T(n) = a\left[aT\left(\frac{n}{b^2}\right) + d\left(\frac{n}{b}\right)\right] + d(n) = a^2T\left(\frac{n}{b^2}\right) + ad\left(\frac{n}{b}\right) + d(n)$$

$$T(n) = a^2\left[aT\left(\frac{n}{b^2}\right) + d\left(\frac{n}{b^2}\right)\right] + ad\left(\frac{n}{b}\right) + d(n)$$

$$= a^3T\left(\frac{n}{b^3}\right) + a^2d\left(\frac{n}{b^2}\right) + ad\left(\frac{n}{b}\right) + d(n)$$

$$= .....$$

$$= a^iT\left(\frac{n}{b^i}\right) + \sum_{j=0}^{i-1} a^jd\left(\frac{n}{b^j}\right)$$

Giả sử n = bk chúng ta được: T(n/bk) = T(1) = 1. Thay vào trên với i = k chúng ta có:

$$T(n) = a^k + \sum_{j=0}^{k-1} a^jd\left(b^{k-j}\right) \quad \text{(I.2)}$$

Trong công thức (I.2), ak = nlogba được gọi là nghiệm thuần nhất (homogeneous solutions).

Nghiệm thuần nhất là nghiệm chính xác khi d(n) = 0 với mọi n. Nói một cách khác, nghiệm thuần nhất biểu diễn thời gian để giải tất cả các bài toán con.

Trong công thức (I.2), $a^k + \sum_{j=1}^{k-1} a^jd\left(b^{k-j}\right)$ được gọi là nghiệm riêng (particular solutions).

Nghiệm riêng biểu diễn thời gian phải trả để tạo ra các bài toán con và tổng hợp các kết quả của chúng. Nhìn vào công thức chúng ta thấy nghiệm riêng phụ thuộc vào hàm tiến triển, số lượng và kích thước các bài toán con.

Khi tìm nghiệm của phương trình (I,1), chúng ta phải tìm nghiệm riêng và so sánh với nghiệm thuần nhất. Nếu nghiệm nào lớn hơn, chúng ta lấy nghiệm đó làm nghiệm của phương trình (I,1).

Để xác định được hàm thuần nhất chúng ta gặp rất nhiều khó khăn, tuy vậy, chúng ta cũng tìm được một lớp các hàm tiến triển có thể dễ dàng xác định nghiệm riêng.

Hàm nhân

Một hàm f(n) được gọi là hàm nhân (multiplicative function) nếu f(m.n) = f(m).f(n) với mọi số nguyên dương m và n.

Ví dụ 5: Hàm f(n) = nk là một hàm nhân, vì f(m.n) = (m.n)k = mk.nk = f(m) f(n)

Nếu d(n) trong (I.1) là một hàm nhân thì theo tính chất của hàm nhân chúng ta có

d(bk-j) = (d(b))k-j và nghiệm riêng của (I.2) là:

$$\sum_{j=0}^{k-1} a^j d\left(b^{k-j}\right) = d(b)^k \sum_{j=0}^{k-1} \left[\frac{a}{d(b)}\right]^j$$

$$= d(b)^k \frac{\left[\frac{a}{d(b)}\right]^k - 1}{\frac{a}{d(b)} - 1} = \frac{a^k - d(b)^k}{\frac{a}{d(b)} - 1} \qquad (I.3)$$

Xét ba trường hợp sau:

- Nếu a > d(b) thì nghiệm riêng là O(ak) = O(nlogba). Như vậy nghiệm riêng và nghiệm thuần nhất bằng nhau do đó T(n) là O(nlogba).

Chúng ta cũng thấy thời gian thực hiện chỉ phụ thuộc vào a, b mà không phụ thuộc vào hàm tiến triển d(n). Vì vậy để cải tiến giải thuật chúng ta cần giảm a hoặc tăng b.

- Nếu a < d(b) thì nghiệm riêng là O(d(b)k) = O(nlogbd(b)). Trong trường hợp này nghiệm riêng lớn hơn nghiệm thuần nhất nên T(n) = O(nlogbd(b)).

Để cải tiến giải thuật chúng ta cần để ý đến cả d(n), a và b cùng mức độ như nhau. Trường hợp đặc biệt quan trọng khi d(n) = nα . Khi đó d(b) = bα và logb(bα) = α. Vì thế nghiệm riêng là O(nα) và do vậy T(n) là O(nα).

- Nếu a = d(b) thì công thức (I.5) không xác đinh nên chúng ta tính trực tiếp nghiệm riêng:

$$\text{Nghiệm riêng} = \sum_{j=0}^{k-1} a^j \left(d(b)\right)^{k-j}$$

$$= d(b)^k \sum_{j=0}^{k-1} \left[\frac{a}{d(b)}\right]$$

$$= d(b)^k \sum_{j=0}^{k-1} 1$$

$$= d(b)^k k$$

$$= n^{\log_b d(b)} \log_b n$$

Vì a= d(b) nên nghiệm riêng là nlogbalogbn và nghiệm này lớn gấp logbn lần nghiệm thuần nhất. Do đó T(n) = O(nlogbalogbn).

Trong trường hợp đặc biệt d(n) = nα chúng ta được T(n) = O(nαlogn).

Chú ý khi giải một phương trình đệ quy cụ thể, chúng ta phải xem phương trình đó có thuộc dạng phương trình tổng quát hay không. Nếu có thì phải xét xem hàm tiến triển có phải là hàm nhân không. Nếu có thì chúng ta xác định a, d(b) và dựa vào sự so sánh giữa a và d(b) mà vận dụng một trong ba trường hợp nói trên.

Ví dụ 6: Giải các phương trình đệ quy sau với T(1) = 1 và

a/- T(n) = 4T(n/2) + n

b/- T(n) = 4T(n/2) + n2

c/- T(n) = 4T(n/2) + n3

Trong mỗi trường hợp, a=4, b=2 và nghiệm thuần nhất là n2. Với d(n) = n chúng ta có d(b) = 2 vì a = 4 > d(b) nên nghiệm riêng cũng là n2 và T(n) = O(n2) trong phương trình (1).

Trong phương trình (3), d(n) = n3, d(b) = 8 và a < d(b). Vì vậy nghiệm riêng là O(nlogbd(b)) = O(n3) và T(n) của (3) là O(n3).

Trong phương trình (2) chúng ta có d(b) = 4 = a nên T(n) = O(n2logn).

Các hàm tiến triển khác

Chúng ta xét hai trường hợp dưới dạng hai ví dụ, trường hợp 1 là tổng quát hóa của hàm bất kỳ là tích của một hàm nhân với một hằng lớn hơn hoặc bằng 1. Trường hợp thứ hai là hàm tiến triển không phải là một hàm nhân.

Ví dụ 7: Giải phương trình đệ quy sau :

T(1) = 1

T(n) = 3T(n/2) + 2n1.5

Ở đây, 2n1.5 không phải là hàm nhân nhưng n1.5 là hàm nhân. Đặt U(n) = T(n)/2 với mọi n thì

U(1) = ½

U(n) = 3U(n/2) + n1.5

Nghiệm thuần nhất khi U(1) = 1 là nlog3 = n1.59; vì U(1) = 1/2 nên nghiệm thuần nhất là n1.59/2 là O(n1.59). Vì a = 3 và b = 2 và b1.5 = 2.82 < a, nghiệm riêng cũng là O(n1.59) và do đó U(n) = O(n1.59) . Vì T(n) = 2U(n) nên T(n) = O(n1.59) hay T(n) = O(nlog3).

Ví dụ 8: Giải phương trình đệ quy sau :

T(1) = 1

T(n) = 2T(n/2) + nlogn

Vì a = b = 2 nên nghiệm thuần nhất là n. Tuy nhiên, d(n) = nlogn không phải là hàm nhân chúng ta phải tính nghiệm riêng bằng cách xét trực tiếp:

$$\sum_{j=0}^{k-1} 2^j 2^{k-j} \log\left(2^{2-j}\right) = 2^k \sum_{j=0}^{k-1} (k-j) = 2^{k-1} k\,(k+1)$$

Vì k = logn chúng ta có nghiệm riêng là O(nlog2n), nghiệm này lớn hơn nghiệm thuần nhất và T(n) = O(nlog2n)

1.4. Bài tập

Bài 1: Tính độ phức tạp của thuật toán sau:

```
void Selection_sort(int a[],int n)
{
 int temp,i,j,m;
 for (i=0;i<=n-2;i++)
 {
  m=i;
  for (j=i+1;j<=n-1;j++)
   if (a[m]<a[j])
   m=j;
  temp=a[i];
  a[i]=a[j];
```

```
   a[j]=temp;
  }
}
```

Bài tập 2: Tính độ phức tạp của thuật toán sau:

```
int ucln(int a,int b)
{
 int r;
 r=a%b;
 while (r!=0)
 {
  a=b;
  b=r;
  r=a%b;
 }
 return b;
}
```

Bài tập 3: Tính độ phức tạp của các đoạn mã lệnh sau:

a) Đoạn mã lệnh 1

```
Sum = 0;
for  (i=1;i<=n;i++)
{
 cin >>x;
Sum = Sum + x;
}
```

b)Đoạn mã lệnh 2:

```
for (i=1;i<=n;i++)
      for (j=1;j<=n;j++)
{
```

```
  c[i][j]=0;
   for (k=1;k<=n;k++)
   c[i][j] = c[i][j] + a[i][k] * b[k][j];
}
```

Bài tập 4: Cho một mảng n số nguyên được sắp thứ tự tăng. Viết hàm tìm một số nguyên trong mảng đó, nếu tìm thấy thì trả về TRUE, ngược lại trả về FALSE. Tính độ phức tạp của thuật toán.

Bài tập 5: Cho đoạn mã lệnh sau:

```
void chuyen(int n, int cot1, int cot2, int cot3)
{
 if (n==1)
    cout<<"chuyen mot dia tu cot " <<cot1 << "sang cot " << cot 3;
 else
   {
    chuyen(n-1, cot1,cot3,cot2);
    chuyen(1,cot1,cot2,cot3);
    chuyen(n-1,cot2,cot1,cot3);
   }
}
```

Hãy tính độ phức tạp của đoạn mã lệnh trên.

Bài tập 6: Xét định nghĩa số tổ hợp chập k của n như sau:

$$C_n^k = \begin{cases} 1 \text{ nếu } k = 0 \text{ hoặc } k = n \\ C_n^{k-1} + C_{n-1}^{k-1} \text{ nếu } 0 < k < n \end{cases}$$

Viết một hàm đệ quy để tính số tổ hợp chập k của n.

Tính thời gian thực hiện của giải thuật nói trên.

CHƯƠNG 2: CÁC KIỂU DỮ LIỆU CƠ SỞ

Mã bài: MH28.2

Mục tiêu:

- Trình bày được khái niệm, phạm vi lưu trữ dữ liệu, các phép xử lý của các kiểu dữ liệu cơ sở như: kiểu số, chuỗi, logic, tập hợp,...;
- Sử dụng được các kiểu dữ liệu cơ sở trong việc mô tả các đối tượng trong các ngôn ngữ lập trình bậc cao như C, Pascal;
- Nghiêm túc, tỉ mỉ, sáng tạo trong việc học và vận dụng vào làm bài tập.

2.1. Các kiểu dữ liệu cơ bản

2.1.1 Kiểu số

2.1.1.1 Kiểu nguyên

Trong C++ cho phép sử dụng các kiểu số nguyên được khai báo bởi từ khóa int, hoặc đi kèm theo int với các từ khóa long, short, unsigned.

Số lượng bit được dùng để lưu trử một giá trị int phụ thuộc vào kích thước từ (word) của máy. Thường thì máy 16-bit sẽ dùng 16 bit để lưu trử một giá trị int , trong khi đó máy 32-bit sẽ dùng 32 bit .

Kích thước và phạm vi biểu diễn của chúng được cho trong bảng sau :

Kiểu	Phạm vi biểu diễn	Kích thước
int	Chiếm 1 từ của máy	
short int , short	-32768 → 32767 ($-2^{15} \rightarrow 2^{15}-1$)	16 bit
unsigned short int	0 → 65535 ($0 \rightarrow 2^{16}-1$)	16 bit
long int , long	-2147483648 → 2147483647 ($-2^{31} \rightarrow 2^{31}-1$)	32 bit
unsigned long int	0 → 4294967295 ($0 \rightarrow 2^{32}-1$)	32 bit
unsigned int	Số nguyên không âm , chiếm 1 từ của máy.	

2.1.1.2 Kiểu số thực

C++ cho phép sử dụng 3 kích thước giá trị thực, tương ứng với 3 từ khóa :

- float
- double
- long double

Kích thước và phạm vi biểu diễn của chúng được cho trong bảng sau :

Kiểu	Ý nghĩa	Phạm vi biểu diễn	Kích thước		Độ chính xác
float	Số thực chính xác đơn	-3.4E+38 → 3.4E+38	32 bit		6 số thập phân
double	Số thực chính xác kép	-1.7E+308 → 1.7E+308,0	64 bit		10 số thập phân
long double		Kích thước 96 bit hoặc 128 bit			

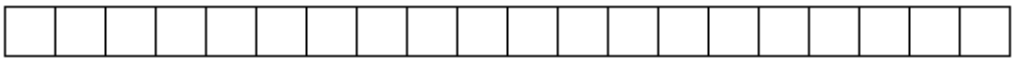
2.1.2 Kiểu ký tự, chuỗi

Trong C++ không có kiểu dữ liệu cơ bản để lưu các xâu kí tự. Để có thể thỏa mãn nhu cầu này, người chúng ta sử dụng mảng có kiểu char. Hãy nhớ rằng kiểu dữ liệu này (char) chỉ có thể lưu trữ một kí tự đơn, bởi vậy nó được dùng để tạo ra xâu của các kí tự đơn.

Ví dụ, mảng sau (hay là xâu kí tự):

char alpha [20];

có thể lưu một xâu kí tự với độ dài cực đại là 20 kí tự. Bạn có thể tưởng tượng tương tự như sau:

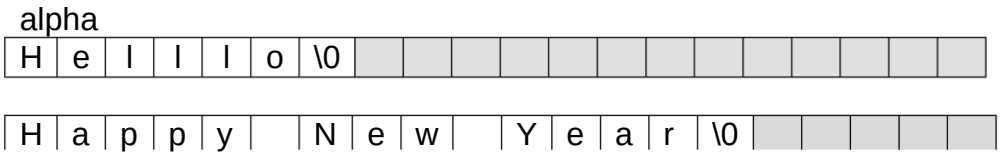


alpha

Kích thước cực đại này không cần phải luôn luôn dùng đến.

Chẳng hạn, alpha có thể lưu xâu "Hello" hay "Merry christmas". Vì các mảng kí tự có thể lưu các xâu kí tự ngắn hơn độ dài của nó, trong C++ đã có một quy ước để kết thúc một nội dung của một xâu kí tự bằng một kí tự null, có thể được viết là '\0'.

Chúng ta có thể biểu diễn alpha (một mảng có 20 phần tử kiểu char) khi lưu trữ xâu kí tự "Hello" và "Happy New Year" theo cách sau:



Chú ý rằng sau nội dung của xâu, một kí tự null ('\0') được dùng để báo hiệu kết thúc xâu. Những ô màu xám biểu diễn những giá trị không xác định.

Khởi tạo các xâu kí tự.

Vì những xâu kí tự là những mảng bình thường nên chúng cũng như các mảng khác. Chẳng hạn, nếu chúng ta muốn khởi tạo một xâu kí tự với những giá trị xác định chúng ta có thể làm điều đó tương tự như với các mảng khác:

char mystring[] = { 'H', 'e', 'l', 'l', 'o', '\0' };

Tuy nhiên, chúng ta có thể khởi tạo giá trị cho một xâu kí tự bằng cách khác: sử dụng các hằng xâu kí tự.

Trong các biểu thức chúng ta đã sử dụng trong các ví dụ trong các chương trước các hằng xâu kí tự để xuất hiện vài lần. Chúng được biểu diễn trong cặp ngoặc kép ("), ví dụ:

"the result is: "

là một hằng xâu kí tự chúng ta sử dụng ở một số chỗ.

Không giống như dấu nháy đơn (') cho phép biểu diễn hằng kí tự, cặp ngoặc kép (") là hằng biểu diễn một chuỗi kí tự liên tiếp, và ở cuối chuỗi một kí tự null ('\0') luôn được tự động thêm vào.

Chúng ta có thể khởi tạo xâu mystring theo một trong hai cách sau đây:

char mystring [] = { 'H', 'e', 'l', 'l', 'o', '\0' };

char mystring [] = "Hello";

Trong cả hai trường hợp mảng (hay xâu kí tự) mystring được khai báo với kích thước 6 kí tự: 5 kí tự biểu diễn Hello cộng với một kí tự null.

Trước khi tiếp tục, tôi cần phải nhắc nhở bạn rằng việc gán nhiều hằng như việc sử dụng dấu ngoặc kép (") chỉ hợp lệ khi khởi tạo mảng, tức là lúc khai báo mảng. Các biểu thức trong chương trình như:

mystring = "Hello";
mystring[] = "Hello";

là không hợp lệ, cả câu lệnh dưới đây cũng vậy:

mystring = { 'H', 'e', 'l', 'l', 'o', '\0' };

Vậy hãy nhớ: Chúng ta chỉ có thể "gán" nhiều hằng cho một mảng vào lúc khởi tạo nó. Nguyên nhân là một thao tác gán (=) không thể nhận vế trái là cả một mảng mà chỉ có thể nhận một trong những phần tử của nó. Vào thời điểm khởi tạo mảng là một trường hợp đặc biệt, vì nó không thực sự là một lệnh gán mặc dù nó sử dụng dấu bằng (=).

Gán giá trị cho xâu kí tự

Vế trái của một lệnh gán chỉ có thể là một phần tử của mảng chứ không thể là cả mảng, chúng ta có thể gán một xâu kí tự cho một mảng kiểu char sử dụng một phương pháp như sau:

mystring[0] = 'H';

mystring[1] = 'e';

mystring[2] = 'l';

mystring[3] = 'l';

mystring[4] = 'o';

mystring[5] = '\0';

Nhưng rõ ràng đây không phải là một phương pháp thực tế. Để gán giá trị cho một xâu kí tự, chúng ta có thể sử dụng loạt hàm kiểu strcpy (string copy), hàm này được định nghĩa trong string.h và có thể được gọi như sau:

strcpy (string1, string2);

Lệnh này copy nội dung của string2 sang string1. string2 có thể là một mảng, con trỏ hay một hằng xâu kí tự, bởi vậy lệnh sau đây là một cách đúng để gán xâu hằng "Hello" cho mystring:

strcpy (mystring, "Hello");

Ví dụ 1:

```
#include <iostream.h>                    J. Soulie
#include <string.h>

int main ()
{
  char szMyName [20];
  strcpy (szMyName,"J. Soulie");
  cout << szMyName;
  return 0;
}
```

Để ý rằng chúng ta phải include file <string.h> để có thể sử dụng hàm strcpy.

Mặc dù chúng ta luôn có thể viết một hàm đơn giản như hàm setstring dưới đây để thực hiện một thao tác giống như strcpy:

Ví dụ 2:

```
// setting value to string                J. Soulie
#include <iostream.h>

void setstring (char szOut [], char szIn
[])
{
  int n=0;
  do {
    szOut[n] = szIn[n];
    n++;
  } while (szIn[n] != 0);
}

int main ()
```

```
{
  char szMyName [20];
  setstring (szMyName,"J. Soulie");
  cout << szMyName;
  return 0;
}
```

Một phương thức thường dùng khác để gán giá trị cho một mảng là sử dụng trực tiếp dòng nhập dữ liệu (cin). Trong trường hợp này giá trị của xâu kí tự được gán bởi người dùng trong quá trình chương trình thực hiện.

Khi cin được sử dụng với các xâu kí tự nó thường được dùng với phương thức getline của nó, phương thức này có thể được gọi như sau:

cin.getline (char buffer[], int length, char delimiter = ' \n');

trong đó buffer (bộ đệm) là địa chỉ nơi sẽ lưu trữ dữ liệu vào (như là một mảng chẳng hạn), length là độ dài cực đại của bộ đệm (kích thước của mảng) và delimiter là kí tự được dùng để kết thúc việc nhập, mặc định - nếu chúng ta không dùng tham số này - sẽ là kí tự xuống dòng ('\n').

Ví dụ sau đây lặp lại tất cả những gì bạn gõ trên bàn phím. Nó rất đơn giản nhưng là một ví dụ cho thấy bạn có thể sử dụng cin.getline với các xâu kí tự như thế nào:

Ví dụ 3:

```
#include <iostream.h>

int main ()
{
  char mybuffer [100];
  cout << "What's your name? ";
  cin.getline (mybuffer,100);
  cout << "Hello " << mybuffer << ".\n";
  cout << "Which is your favourite
```

```
What's your name? Juan
Hello Juan.
Which is your favourite team? Inter Milan
I like Inter Milan too.
```

```
team? ";
  cin.getline (mybuffer,100);
  cout << "I like " << mybuffer << "
too.\n";
  return 0;
}
```

Chú ý trong cả hai lời gọi cin.getline chúng ta sử dụng cùng một biến xâu (mybuffer). Những gì chương trình làm trong lời gọi thứ hai đơn giản là thay thế nội dung của buffer trong lời gọi cũ bằng nội dung mới.

Nếu bạn còn nhớ phần nói về giao tiếp với, bạn sẽ nhớ rằng chúng ta đã sử dụng toán tử >> để nhận dữ liệu trực tiếp từ đầu vào chuẩn. Phương thức này có thể được dùng với các xâu kí tự thay cho cin.getline. Ví dụ, trong chươn trình của chúng ta, khi chúng ta muốn nhận dữ liệu từ người dùng chúng ta có thể viết:

cin >> mybuffer;

lệnh này sẽ làm việc như nó có những hạn chế sau mà cin.getline không có:

Nó chỉ có thể nhận những từ đơn (không nhận được cả câu) vì phương thức này sử dụng kí tự trống(bao gồm cả dấu cách, dấu tab và dấu xuống dòng) làm dấu hiệu kết thúc..

Nó không cho phép chỉ định kích thước cho bộ đệm. Chương trình của bạn có thể chạy không ổn định nếu dữ liệu vào lớn hơn kích cỡ của mảng chứa nó.

Vì những nguyên nhân trên, khi muốn nhập vào các xâu kí tự bạn nên sử dụng cin.getline thay vì cin >>.

Chuyển đổi xâu kí tự sang các kiểu khác.

Vì một xâu kí tự có thể biểu diễn nhiều kiểu dữ liệu khác như dạng số nên việc chuyển đổi nội dung như vậy sang dạng số là rất hữu ích. Ví dụ, một xâu có thể mang giá trị "1977"nhưng đó là một chuỗi gồm 5 kí tự (kể cả kí tự null) và không dễ gì chuyển thành một số nguyên. Vì vậy thư viện cstdlib (stdlib.h) đã cung cấp 3 macro/hàm hữu ích sau:

atoi: chuyển xâu thành kiểu int.

atol: chuyển xâu thành kiểu long.

atof: chuyển xâu thành kiểu float.

Tất cả các hàm này nhận một tham số và trả về giá trị số (int, long hoặc float). Các hàm này khi kết hợp với phương thức getline của cin là một cách đáng tin cậy hơn phương thức cin>> cổ điển khi yêu cầu người sử dụng nhập vào một số.

Ví dụ 4:

```
#include <iostream.h>
#include <stdlib.h>

int main ()
{
  char mybuffer [100];
  float price;
  int quantity;
  cout << "Enter price: ";
  cin.getline (mybuffer,100);
  price = atof (mybuffer);
  cout << "Enter quantity: ";
  cin.getline (mybuffer,100);
  quantity = atoi (mybuffer);
  cout << "Total price: " <<
price*quantity;
  return 0;
}
```

```
Enter price: 2.75
Enter quantity: 21
Total price: 57.75
```

Các hàm để thao tác trên chuỗi

Thư viện cstring (string.h) không chỉ có hàm strcpy mà còn có nhiều hàm khác để thao tác trên chuỗi. Dưới đây là giới thiệu lướt qua của các hàm thông dụng nhất:

strcat: char* strcat (char* dest, const char* src);

Gắn thêm chuỗi src vào phía cuối của dest. Trả về dest.

strcmp: int strcmp (const char* string1, const char* string2);

So sánh hai xâu string1 và string2. Trả về 0 nếu hai xâu là bằng nhau.

strcpy: char* strcpy (char* dest, const char* src);

Copy nội dung của src cho dest. Trả về dest.

strlen: size_t strlen (const char* string);

Trả về độ dài của string.

Chú ý: char* hoàn toàn tương đương với char[]

Ví dụ 5: Chương trình sau sẽ hiển thị một dòng chữ chạy trên màn hình.

```
#include <stdio.h>
void main()
{
  char qcao[81], i=0, len;
  textattr(0x1e);
  printf("\nNhap vao dong quang cao : ");
  gets(qcao);
  len = strlen(qcao);
  while (!kbhit())
  {
    movetext(2, 1, 80, 1, 1, 1);
    gotoxy(80, 1);
    cprintf("%c", qcao[i++]);
    delay(100);
    i %= len;
  }
  getch();
}
```

Ví dụ 6: Chương trình sau dùng các hàm dựng sẵn của C++ để chuyển sang các dạng ký tự khác nhau.

```
#include"stdio.h"
#include"alloc.h"
#include"string.h"
#include"conio.h"
```

```
char *thaythe(unsigned char kytu,size_t lan)
   {
          char *inkytu;
          inkytu=calloc(lan,sizeof(char)+1);
          memset(inkytu,kytu,lan);
          return(inkytu);
   }
main()
   {
          char *s, *t;
          s=calloc(60,sizeof(char));
          t=calloc(60,sizeof(char));
          clrscr();
          textcolor(YELLOW);
          cprintf(" - Nhap chuoi thu nhat (chu thuong)= ");
          gets(s);
          textcolor(LIGHTRED);
          cprintf(" - Nhap chuoi thu hai (chu thuong) = ");
          gets(t);
          cprintf("\n %s",thaythe(205,60));
          textcolor(LIGHTCYAN);
          cprintf("\n\r %10c DOI RA CHU HOA",' ');
          cprintf("\n\n\r");
          puts(strupr(s));
          cprintf("\r");
          puts(strupr(t));
          cprintf("\n %s",thaythe(205,60));
          textcolor(LIGHTMAGENTA);
          cprintf("\n\r %10c DOI TRO LAI CHU THUONG",' ');
          cprintf("\n\n\r");
```

```
        puts(strlwr(s));
        cprintf("\r");
        puts(strlwr(t));
        cprintf("\n %s",thaythe(205,60));
        textcolor(LIGHTGREEN);
        cprintf("\n\r %10c DAO NGUOC CHU",' ');
        cprintf("\n\n\r");
        puts(strrev(s));
        cprintf("\r");
        puts(strrev(t));
        cprintf("\n %s",thaythe(205,60));
        getch();
   }
```

Ví dụ 7:Chương trình sau thực hiện xóa một số ký tự

```
#include"conio.h"
#include"alloc.h"
void xoa(char *nguon, char vitri, char so)
   {
        char k;
        for (k=vitri-1;nguon[k+so];k++)
             nguon[k]=nguon[k+so];
             nguon[k]='\0';
   }
void main()
   {
        int m,n;
        char *s;
        s=calloc(100,sizeof(char));
        clrscr();
        textcolor(YELLOW);
```

```
        cprintf("\n - Nhap vao mot chuoi :");
        gets(s);
        textcolor(LIGHTRED);
        cprintf("\r - Muon xoa tu vi tri nao : ");
        scanf("%d",&m);
        textcolor(LIGHTMAGENTA);
        cprintf("\r- Muon xoa bao nhieu ky tu : ");
        scanf("%d",&n);
        textcolor(LIGHTRED);
        cprintf("\n\r**********************************");
        textcolor(YELLOW);
        cprintf("\n\n\r     + Chuoi nguon la :");
        textcolor(LIGHTCYAN);
        cprintf("\n\r %s",s);
        textcolor(LIGHTGREEN);
        cprintf("\n\r     + Chuoi nguon dai :%d ky tu",strlen(s));
        textcolor(LIGHTRED);
        cprintf("\n\r     + Sau khi xoa con lai chuoi : ");
        xoa(s,m,n);
        textcolor(LIGHTMAGENTA);
        cprintf("\n\r %s",s);
        textcolor(LIGHTGREEN);
        cprintf("\n\r     + Chieu dai la : %d ky tu",strlen(s));
        textcolor(LIGHTCYAN);
        cprintf("\n\n\r * Bam phim bat ky de ket thuc");
        getch();
    }
```

Ví dụ 8: Chương trình sau thực hiện đảo ngược chuỗi ký tự.

```
#include"conio.h"
#include"stdio.h"
```

```
#include"string.h"
#include"alloc.h"
void main()
   {
        char *chuoi;
        chuoi=(char *)calloc(80,sizeof(char));
        clrscr();
        cprintf("\n- Nhap vao 1 chuoi : ");
        gets(chuoi);
        cprintf("\n\r- Chuoi nay co : %d ky tu ke ca ky tu trong \n",strlen(chuoi));
        cprintf("\n\r- Dao nguoc chuoi nay thanh \n ");
        cprintf("\n\r %s ",strrev(chuoi));
        cprintf("\n\n\r * Bam phim bat ky de ket thuc");
        getch();
   }
```

Kết quả thực thi chương trình:

- Nhap vao 1 chuoi : Khoa Cong Nghe Thong Tin

- Chuoi nay co : 13 ky tu ke ca ky tu trong

- Dao nguoc chuoi nay thanh

niT gnohT ehnN gnoC aohK

* Bam phim bat ky de ket thuc

Ví dụ 9: Chương trình sau minh họa hàm cộng chuỗi.

```
#include"string.h"
#include"stdio.h"
#include <conio.h>
```

```
void main()
   {
         char mot[100]="Que huong la chum khe ngot";
         char hai[100]=" Cho con treo hai moi ngay";
         char ba[100]="Que huong la duong di hoc";
         char bon[100]=" Con ve rop buom vang bay";
         clrscr();
         printf("\nCau \" %s\". Co chieu dai :%d ky tu ",mot,strlen(mot));
         printf("\nCau \" %s\". Co chieu dai :%d ky tu ",hai,strlen(hai));
         printf("\nCau \" %s\". Co chieu dai :%d ky tu ",ba,strlen(ba));
         printf("\nCau \" %s\". Co chieu dai :%d ky tu ",bon,strlen(bon));
         strcat(mot,hai);
         strcat(ba,bon);
         printf("\n");
         printf("\n PHEP CONG CHUOI KY TU \n");
         printf("\n + Dem cau 1 + cau 2");
         printf("\nHai cau %s",mot);
         printf("\n - Co chieu dai: %d ky tu :",strlen(mot));
         printf("\n");
         printf("\n + Dem cau 3 + cau 4");
         printf("\nHai cau %s",ba);
         printf("\n - Co chieu dai : %d ky tu",strlen(ba));
         getch();
   }
```

Kết quả thực thi chương trình:

Cau " Que huong la chum khe ngot". Co chieu dai :26 ky tu

Cau " Cho con treo hai moi ngay". Co chieu dai :26 ky tu

Cau " Que huong la duong di hoc". Co chieu dai :25 ky tu

Cau " Con ve rop buom vang bay". Co chieu dai :25 ky tu

PHEP CONG CHUOI KY TU

+ Dem cau 1 + cau 2

Hai cau Que huong la chum khe ngot Cho con treo hai moi ngay

- Co chieu dai: 52 ky tu :

+ Dem cau 3 + cau 4

Hai cau Que huong la duong di hoc Con ve rop buom vang bay

- Co chieu dai : 50 ky tu

2.2 Kiểu dữ liệu có cấu trúc

2.2. 1 Kiểu dữ liệu con trỏ

Kiểu dữ liệu con trỏ là kiểu dữ liệu lưu trữ địa chỉ của một ô nhớ trong máy tính. Biến con trỏ là biến lưu trữ địa chỉ một biến khác hay lưu địa chỉ của một ô nhớ trong máy tính.

Để khai báo biến con trỏ, chúng ta phải khai báo theo khuôn dạng sau:

Tên_kiểu_dữ_liệu *tên_biến_con_trỏ;

Để lấy địa chỉ của một biến chúng ta phải sử dụng cú pháp sau:

Tên_biến_con_trỏ = &tên_biến;

Ví dụ sau cho thấy việc khai báo và sử dụng biến con trỏ:

```
int a;
int *p; //biến p là biến con trỏ
p=&a; //biến p trỏ đến biến a
*p = 5 ; lệnh này tương đương với a = 5
```

Trong ví dụ trên, biến con trỏ p lưu trữ địa chỉ của biến a trong bộ nhớ.

Tuy nhiên, con trỏ thể hiện trong phần này chưa nói lên hết khả năng mạnh mẽ của kiểu dữ liệu con trỏ. Để sử dụng hiệu quả con trỏ cũng như các bộ nhớ hiện có trên hệ thống, chúng ta sẽ nghiên cứu ở các bài sau.

2.2.2 Kiểu cấu trúc

Cấu trúc là một kiểu dữ liệu được người dùng tự định nghĩa, kiểu dữ liệu này chứa trong nó các phần tử mà mỗi phần tử có thể có kiểu dữ liệu khác nhau. Kiểu cấu trúc cho phép mô tả các đối tượng có cấu trúc phức tạp.

Khai báo tổng quát của kiểu cấu trúc (struct) như sau:

```
struct <tên kiểu cấu trúc>
{
 <kiểu dữ liệu 1>   <thành phần 1>;
 <kiểu dữ liệu 2>   <thành phần 2>;
 <kiểu dữ liệu 3>   <thành phần 3>;
  ……
 <kiểu dữ liệu n>   <thành phần n>
};
```

Ví dụ 10: Để mô tả các thông tin về một con người, chúng ta có thể khai báo một kiểu dữ liệu và khai báo một cấu trúc như sau:

```
struct  nguoi
{
  char hoten[35];
  int namsinh;
  char noisinh[60];
  int gioitinh; {0- nu;1:nam}
  char diachi[79];
}
```

Kiểu cấu trúc được tổ chức trong bộ nhớ với một cấu trúc liên tục, không chồng lấp lên nhau:

35 bytes	2 bytes	60 bytes	2 bytes	79b ytes
Hot en	nam sinh	Noi sinh	gioi tinh	dia chi

Kiểu cấu trúc bổ sung những hạn chế của kiểu mảng, giúp chúng ta có khả năng thể hiện các đối tượng đa dạng của thế giới thực vào trong máy tính một cách dễ dàng và chính xác hơn.

Khi khai báo biến có kiểu dữ liệu cấu trúc, chúng ta truy xuất đến các thành phần con bên trong bằng dấu chấm (.).

Ví dụ 14: Nhập danh sách các học sinh có kiểu dữ liệu nguoi như trên. In ra màn hình thông tin của các học sinh có năm sinh bé hơn 1985.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
void main()
{
 struct nguoi
 {
  char hoten[35];
  int namsinh;
  char noisinh[60];
  int gioitinh;
  char diachi[79];
 };
 nguoi ds[100];
 int n,i;
 clrscr();
 cout<<"Nhap vao so hoc sinh:";cin>>n;
 cout<<"Nhap thong tin cho tung hoc sinh:"<<endl;
 for (i=0;i<=n-1;i++)
  {
   cout<<"Nhap thong tin cho hoc sinh thu "<<i+1<<endl;
   cout<<"Ho ten:";gets(ds[i].hoten);//scanf("%s",L[i].hoten);
   cout<<"Nam sinh:";cin>>ds[i].namsinh;
   cout<<"Noi sinh:";gets(ds[i].noisinh);
   cout<<"Gioi tinh(1-Nam;0-Nu):";cin>>ds[i].gioitinh;
   cout<<"Dia chi:";gets(ds[i].diachi);
  }
 cout<<"Thong tin ve cac hoc sinh co nam sinh be hon 1985"<<endl;
```

```
 for (i=0;i<=n-1;i++)
  if (ds[i].namsinh<1985)
   {
    cout<<"Thong tin:"<<endl;
    cout<<"Ho ten:"<<ds[i].hoten<<endl;
    cout<<"nam sinh:"<<ds[i].namsinh<<endl;
    cout<<"Noi sinh:"<<ds[i].noisinh<<endl;
    cout<<"Gioi tinh:"<<ds[i].gioitinh<<endl;
    cout<<"Dia chi:"<<ds[i].diachi<<endl;
   }
 getch();
}
```

Kết quả thực thi chương trình minh họa như sau:

```
Nhap vao so hoc sinh:2
Nhap thong tin cho tung hoc sinh:
Nhap thong tin cho hoc sinh thu 1
Ho ten:Nguyen Van A
Nam sinh:1975
Noi sinh:Lam Dong
Gioi tinh(1-Nam;0-Nu):1
Dia chi:01 Hoang Van Thu
Nhap thong tin cho hoc sinh thu 2
Ho ten:Le Thi Tam
Nam sinh:1985
Noi sinh:Duc Trong
Gioi tinh(1-Nam;0-Nu):0
Dia chi:Hiep An
Thong tin ve cac hoc sinh co nam sinh be hon 1985
Thong tin:
Ho ten:Nguyen Van A
```

nam sinh:1975

Noi sinh:Lam Dong

Gioi tinh:1

Dia chi:01 Hoang Van Thu

Ví dụ 11: Nhập danh sách có tối đa 100 nhân viên gồm 2 thông tin: họ tên và tuổi. Tính tuổi trung bình của các nhân viên trong công ty và in ra danh sách các nhân viên có tuổi lớn hơn tuổi trung bình.

```
#include <conio.h>
#include <stdio.h>
#include <iostream.h>
void main()
{
 struct nv
 {
  char hoten[35];
  int tuoi;
 };
 nv nhanvien[100];
 float tuoitrungbinh=0;
 int n,i;
 clrscr();
 cout<<"Nhap so nhan vien:";cin>>n;
 for (i=0;i<=n-1;i++)
 {
  cout<<"Nhap thong tin cho nhan vien thu "<<i+1<<endl;
  cout<<"Ho ten:";gets(nhanvien[i].hoten);
  cout<<"Tuoi:";cin>>nhanvien[i].tuoi;
 }
 for (i=0;i<=n-1;i++)
  tuoitrungbinh+=float(nhanvien[i].tuoi)/float(n);
```

```
 cout<<"Tuoi trung binh cua cong ty:"<<tuoitrungbinh<<endl;
 cout<<"Danh sach cac nhan vien co tuoi lon hon tuoi trung binh:"<<endl;
 for (i=0;i<=n-1;i++)
  if (nhanvien[i].tuoi>tuoitrungbinh)
    {
     cout<<"Thong tin"<<endl;
     cout<<nhanvien[i].hoten<<endl;
     cout<<nhanvien[i].tuoi<<endl;
    }
 getch();
}
```

Kết quả thực thi chương trình được minh họa:

```
Nhap so nhan vien:3
Nhap thong tin cho nhan vien thu 1
Ho ten:Nguyen Van A
Tuoi:26
Nhap thong tin cho nhan vien thu 2
Ho ten:Nguyen Tan Beo
Tuoi:40
Nhap thong tin cho nhan vien thu 3
Ho ten:Nguyen Van Teo
Tuoi:20
Tuoi trung binh cua cong ty:28.666666
Danh sach cac nhan vien co tuoi lon hon tuoi trung binh:
Thong tin
Nguyen Tan Beo
40
```

Ví dụ 12: Viết chương trình tạo một mảng các điểm ngẫu nhiên (dùng cấu trúc điểm). Kiểm tra xem các điểm phát sinh này có thỏa mãn 2x+4y>20. Nhập r là bán

kính của một đường tròn tâm 0. Kiểm tra xem các điểm phát sinh ở trên có nằm trên đường tròn này hay không? In kết quả lên màn hình.

```
#include"stdio.h"
#include"stdlib.h"
#include”conio.h”
#define Max 100
typedef struct
   {
        int x,y;
   }toa_do;
toa_do M[Max];
unsigned char k;
int r;
char tim_thay;
void khoi_tao()
   {
        randomize();
        for (k=0; k<Max; k++)
             {
                  M[k].x = random(20)-10;
                  M[k].y = random(20)-10;
             }
   }
void tim_kiem1()
   {
        tim_thay=0;
        puts("Cac diem tren mat phang thoa 2x + 4y > 20 :");
        for (k=0; k<Max; k++)
             if (2*M[k].x + 4 * M[k].y > 20)
                  {
```

```
                        tim_thay = 1;
                        printf("( %2d\, %2d )",M[k].x, M[k].y);
                    }
            if (tim_thay)
                    puts("\n..la cac diem thoa x + 2y > 3");
            else
                    puts("\nKhong co diem nao nhu vay");
    }
void tim_kiem2()
    {
        tim_thay=0;
        printf("-Nhap ban kinh R= ");
        scanf("%d",&r);
        printf("Cac diem tren vong tron tam O ban kinh %d :\n",r);
        for (k=0; k<Max; k++)
            if (M[k].x * M[k].x + M[k].y * M[k].y == r*r)
                {
                    tim_thay=1;
                    printf("( %2d\, %2d )",M[k].x, M[k].y);
                }
            if (tim_thay)
            printf("\n..la cac diem tren vong tron tam O ban kinh %d",r);
            else
                    puts("\nKhong co diem nao nhu vay");
    }
void main()
    {
        khoi_tao();
        tim_kiem1();
        tim_kiem2();
```

```
        printf("\n\t Bam phim bat ky de ket thuc");
        getch();
    }
```

Kết quả một lần chạy chương trình:

```
Cac diem tren mat phang thoa 2x + 4y > 20 :
(  1,  7 )( -4,  8 )(  5,  7 )(  8,  4 )( -1,  9 )( -7,  9 )(  6,  5 )(  0,  9 )
(  2,  6 )(  2,  9 )( -3,  9 )(  3,  9 )(  9,  5 )( -3,  9 )(  1,  8 )(  9,  5 )
(  7,  4 )(  6,  6 )( -5,  8 )(  9,  6 )(  9,  4 )
..la cac diem thoa x + 2y > 3
-Nhap ban kinh R= 10
Cac diem tren vong tron tam O ban kinh 10 :
(  8, -6 )( -10,  0 )(  6, -8 )
..la cac diem tren vong tron tam O ban kinh 10
      Bam phim bat ky de ket thuc
```

2.3. Kiểu hợp

Kiểu hợp khá giống kiểu cấu trúc. Tuy nhiên, tất cả các thành phần được lưu trữ trên cùng một địa chỉ bộ nhớ. Ví dụ:

```
union viduhop
{
 long lonnhat;
 int thap;
};
```

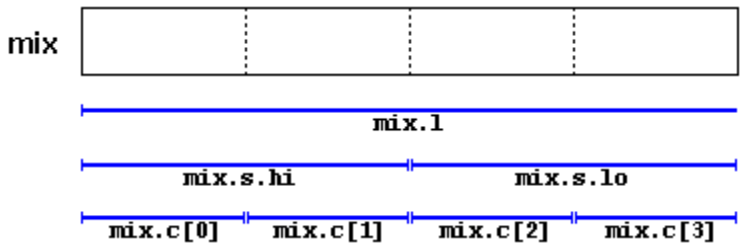
Biến a có kiểu dữ liệu viduhop và thực hiện phép gán: a.lonnhat bểu diễn trong bộ nhớ như sau:

a.lonnhat=502010416	
2bytes	2bytes
11101111011000001001000110000	
a.thap	

Một trong những công dụng của union là dùng để kết hợp một kiểu dữ liêu cơ bản với một mảng hay các cấu trúc gồm các phần tử nhỏ hơn. Ví dụ:

```
union mix_t{
  long l;
  struct {
    short hi;
    short lo;
    } s;
  char c[4];
} mix;
```

định nghĩa ba phần tử cho phép chúng ta truy xuất đến cùng một nhóm 4 byte: mix.l, mix.s và mix.c mà chúng ta có thể sử dụng tuỳ theo việc chúng ta muốn truy xuất đến nhóm 4 byte này như thế nào. Tôi dùng nhiều kiểu dữ liệu khác nhau, mảng và cấu trúc trong union để bạn có thể thấy các cách khác nhau mà chúng ta có thể truy xuất dữ liệu.



Kiểu hợp khá thuận tiện trong việc xử lý các chương trình tính theo byte trong một luồng dữ liệu nhiều byte nhận được, chẳng hạn, xử lý các dữ liệu từ các cổng điều khiển.

Ngoài ra, kiểu hợp còn rất thuận lợi trong việc xử lý số liệu đến bit, byte hay xử lý các giá trị ở mức độ hệ thống.

Ví dụ 14: Chương trình sau hiển thị giá trị byte thấp và byte cao của một số nguyên int.

```
#include <stdio.h>
#include <iostream.h>
#include <conio.h>
union kieu_hop        {
            int i;
```

```
        char ch[2];
    };
kieu_hop uni;
void hien_thi(kieu_hop so)
    {
        clrscr();
        cout<<"\n\n    CHUONG TRINH MINH HOA SU DUNG UNION\n";
        printf("-Hien thi tri 16 (thap luc) cua Byte thap : %X\n",so.ch[0]);
        printf("-Hien thi tri 16 (thap luc) cua Byte cao  : %X\n",so.ch[1]);
        printf("\n       Bam phim bat ky de ket thuc");
        getch();
    }

void main()
    {
        uni.i=0X2040;
        hien_thi(uni);
    }
```

Kết quả thực thi chương trình:

```
   CHUONG TRINH MINH HOA SU DUNG UNION
-Hien thi tri 16 (thap luc) cua Byte thap : 40
-Hien thi tri 16 (thap luc) cua Byte cao  : 20

      Bam phim bat ky de ket thuc
```

Ví dụ 15: Hiển thị bảng mã ASCII theo hai kiểu mã: thập phân và nhị phân.

```
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
typedef struct
```

```
    {
        unsigned a: 1;
        unsigned b: 1;
        unsigned c: 1;
        unsigned d: 1;
        unsigned e: 1;
        unsigned f: 1;
        unsigned g: 1;
        unsigned h: 1;
    }kytu;
union ma
    {
        char ch;
        kytu b;
    };
void hien_thi(char c)
    {
        ma u;
        u.ch = c;
        printf("\n %c   %3d %u %u %u %u %u %u %u ",
    u.ch,u.ch,u.b.h,u.b.g,u.b.f,u.b.e,u.b.d,u.b.c,u.b.b,u.b.a);
    }
void main()
    {
        char c;
        clrscr();
        printf("\n            B A N G   M A   A S C I I \n");
    cout<<"In theo khung dang: Ky tu  Thap phan Nhi phan"<<endl;
        for (c=1; c<7; hien_thi(c++));
            for (c='A'; c<'K'; hien_thi(c++));
```

```
            printf("\n
**************************************************");
            printf("\n          Bam phim bat ky de ket thuc");
            getch();
    }
```

Chạy chương trình ta đạt được kết quả:

B A N G M A A S C I I

In theo khung dang: Ky tu Thap phan Nhi phan

☺ 1 0 0 0 0 0 0 0 1

☻ 2 0 0 0 0 0 0 1 0

♥ 3 0 0 0 0 0 0 1 1

♦ 4 0 0 0 0 0 1 0 0

♣ 5 0 0 0 0 0 1 0 1

♠ 6 0 0 0 0 0 1 1 0

A 65 0 1 0 0 0 0 0 1

B 66 0 1 0 0 0 0 1 0

C 67 0 1 0 0 0 0 1 1

D 68 0 1 0 0 0 1 0 0

E 69 0 1 0 0 0 1 0 1

F 70 0 1 0 0 0 1 1 0

G 71 0 1 0 0 0 1 1 1

H 72 0 1 0 0 1 0 0 0

I 73 0 1 0 0 1 0 0 1

J 74 0 1 0 0 1 0 1 0

****

Bam phim bat ky de ket thuc

2.4 Kiểu tập tin (file)

Kiểu tập tin là kiểu dữ liệu được lưu trữ trên đĩa với một tên tập tin xác định.

- File có 2 loại :

+ Text file (file văn bản).

+ Binary (nhị phân : dbf, doc, bitmap,...).

- File văn bản chỉ khác binary khi xử lý ký tự chuyển dòng (LF) (mã 10) được chuyển thành 2 ký tự CR (mã 13) và LF (mã 10) và khi đọc 2 ký tự liên tiếp CR và LF trên file cho chúng ta một ký tự LF.

- Các thao tác trên file thực hiện thông qua con trỏ kiểu FILE. Mỗi biến FILE có 1 con trỏ lúc đầu sẽ trỏ vào phần tử đầu tiên của file. Sau mỗi thao tác đọc hay ghi dữ liệu con trỏ tự động dời xuống mẫu tin kế tiếp. Làm việc trên kiểu File thường có 3 công đoạn : mở file, nhập xuất thông trên file và đóng file.

* Một số hàm thông dụng thao tác trên file (tập tin/tệp tin) :

+ Mở file : FILE *fopen (char *filename, char *mode); Nếu có lỗi fp sẽ trỏ đến NULL.

+ Các chế độ mở file :

" r" " rt " / " rb " : mở file để đọc theo kiểu văn bản / nhị phân - file phải tồn tại trước nếu không sẽ có lỗi.

"w" "wt" / " wb " : mở (tạo) file mới để ghi theo kiểu văn bản/nhị phân - nếu file đã có nó sẽ bị xóa (ghi đè)(luôn luôn tạo mới).

"a" "at"/ "ab" : mở file để ghi bổ sung (append) thêm theo kiểu văn bản hoặc nhị phân(chưa có thì tạo mới).

+ Đóng file : int fclose (file + biến file) ;

Ví dụ 15: Minh họa mở tập tin.

```
void main ( )
{
 FILE *fp ;
 fp = fopen ("c:\\THUCTAP\\Data.txt", "wt" );
 if (fp = NULL )
  printf ( " không mở được file c:\Thuctap\data.txt");
else
 {
   < xử lý file >
```

```
 }
 fclose (fp) ; /* đóng file */
}
```

+ Đóng tất cả các tập đang mở : int fcloseall(void) ; nếu thành công trả về số nguyên bằng tổng số các file đóng được, ngược lại trả về EOF.

+ Hàm xóa tập : remove (const + char*ten tập) ; nếu thành công cho giá trị 0, ngược lại EOF.

+ Hàm kiểm tra cuối tập : int feof(FILE*fp) : !=0 : nếu cuối tập= 0 : chưa cuối tập.

+ Hàm int putc (int ch, FILE*fp);

Hàm int fputc(int ch, FILE*fp);

Công dụng của hai hàm này :ghi một ký tự lên tập fp theo khuôn dạng được xác định trong chuỗi điều khiển dk. Chuỗi dk và danh sách đối tương tự hàm printf( ).
+ Hàm int fscanf (FILE *fp, const char *dk, ...);

Công dụng : đọc dữ liệu từ tập tin fp theo khuôn dạng (đặc tả) làm việc giống scanf( ).

Ví dụ 16: Giả sử có file c:\data.txt lưu 10 số nguyên 1 5 7 9 8 0 4 3 15 20 . Hãy đọc các số nguyên thêm vào một mãng sau đó sắp xếp tăng dần rồi ghi vào file datasx.txt.

```
#include <stdio.h>
#include<conio.h>
#include<stdlib.h>
#define n 10
void main ( )
{
 FILE *fp ;
 int i, j, t, a[n];
 clrscr ( ) ;
 fp = fopen (" c :\\data.txt ", "rt" );
/* mở file để đọc vào mãng */
```

```
      if (fp = NULL)
      {
        printf ("không mở được file ");
         exit (1);
      }
     while (1)
     { fscanf (fp,"%d",&a[i] ;
      i++;
  if (foef(fp) )
       break ;
     /* Sắp xếp mảng */
         for ( i=0 ; i<n-1 ; i++)
         for (j=i+1; j<n ; j++)
           if (a[i]<a[j] )
               { t = a[i] ; a[i]=a[j] ; a[j] = t ; }
        fclose (fp);
     /* mở file datasx.txt để ghi sau khi sắp xếp */
         fp = fopen ("c:\\datasx.txt ", "wt");
       for ( i=0 ; i<n;i++)
         printf (fp,"%2d", a[i] );
          fclose (fp);
        i = 0 ;
     while (1)
     { fscanf (fp,"%d",&a[i] ;
      i++;
  if (foef(fp) )
       break ;

}
```

- Hàm int fputs (const char *s, file *fp)

Công dụng : ghi chuỗi s lên tập tin fp (dấu "\0" ghi lên tập) nếu có lỗi hàm cho eof.

- Hàm char fgets (char *s, int n , FILE *fp);

Công dụng : đọc 1 chuỗi ký tự từ tập tin fp chứa vào vùng nhớ s. Việc đọc kết thúc khi : hoặc đã đọc n-1 ký tự hoặc gặp dấu xuống dòng(cắpmã 13 10). khi đó mã 10 được đưa vào chuỗi kết quả.

- Hàm int fwrite (void *p, int size , int n , FILE*fp);

p : là con trỏ trỏ tới vùng nhớ chứa dữ liệu cần ghi.

size : là kích thước của mẫu tin theo byte.

n số mẫu tin cần ghi.

fp là con trỏ tập.

chẳng hạn, fwrite(&tam) size of(tam),1,fv); /* tam là 1 mẫu tin(record) nào đó*/ Công dụng : ghi một mẫu tin (record) kích thước sizebyte (size of (tam)) từ vùng nhớ p(&tam) lên tập fp. Hàm sẽ trả về một giá trị = số mẫu tin thực sự ghi được.

+ Hàm int fread (void*p), int size , int n, FILE *fp);

p : là con trỏ trỏ tới vùng nhớ chứa dữ liệu đọc được.

size là kích thước của mẫu tin theo byte

n : là số mẫu tin cần đọc, fp là con trỏ tập tin.

Chẳng hạn, fread (&tam, size of(KIEUHS) , 1, 4)>0)

Công dụng : đọc n(1) mẫu tin kích thước sizebyte (size of(tam)) từ tập tin fp chứa vào vùng nhớ p(&tam). Hàm trả về một giá trị bằng số mẫu tin thực sự đọc được.

Ví dụ17: Nhập vào danh sách lớp gồm n học viên ("nhập vào). Thông tin về mỗi học viên gồm Họ tên, phái , điểm, kết quả. Xét kết quả theo điều kiện sau : nếu Điểm>= 5 (đậu), điểm <5 : rớt. Sau đó sắp xếp theo điểm và ghi vào tập tin c:\lop.txt. Đọc lại tập tin c:\lop.txt và xét lại kết quả nếu điểm =4 và phái là nữ sẽ đậu và chép sang tập tin c:\ketqua.txt.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
struct
```

```
{ char ten[20] ; char phai[4] ; int diem ; char kq[4] ; } KieuHV;
KieuHV lop[100] , tam;
/* Hàm nhập danh sách n học viên */
void nhapds ( int n, KieuHV lop[ ] )
{ int i , diem ;
for ( i=0; i<n ; i++ )
{
printf (" nhập họ và tên người thứ %d : " , i + 1) ;
gets ( lop[i].ten);
printf ("phái (nam/nữ ) : ") ; gets (lop[i].phai );
printf ("nhập điểm = ") ; scanf ("%d%c*c", &diem); lop[i].diem=diem;
if (diem>5)
strcpy (lop[i].kq,"Đậu");
else
  strcpy (lop[i].kq, "rớt " ) ;
}
/* Hàm sắp xếp */
void sapxep ( int n , KieuHV lop[ ] )
{ int i , j ;
 KieuHV tam;
for ( i=0 ; i<n-1; i++)
for ( j=i+1 ; j<0; j++)
if (lop[i].diem< lop[j]diem )
{ tam = lop[i] ; lop[i] = lop[j] ; lop [i] = tam ;}
}
/* Hàm in danh sách */
void inds ( int n, KieuHS lop[ ] )
{ int i ;
for ( i=0 ; i<n ; i++)
printf ("\n %20s %5s%5d%5s, lop[i].ten, lop[i].phai, lop[i].diem, LOP[I].kq );
```

```
void main ( )
    { int i , j, n, t, diem ; FILE *fp, *fr ;
     printf ("\n nhập sĩ số : ") ; scanf("%d%*c",&n);
     lop = (KieuHV*) malloc (n*size of (KieuHV));
     nhapds(n, lop) ; sapxep ( n, lop ); inds( n, lop); getch( );
    fp = fopen ( "c :\\lop.txt ", "wb");
    for ( i = 0; i<n ; i++)
    fwrite (&lop[ i], size of (KieuHV), 1, fp);
     fclose(fp);
    printf ("\n ghi dữ liệu xong ");
    printf("\n in file sau khi sắp xếp và xét kết quả lại ");
    fr = fopen ("c:\\ketqua.txt", "wb");
    while ( fread (&tam, size of ( KieuHV), 1, fp ) > 0)
    {
    printf ("\n %s %s%d%s", tam.ten, tam.phai, tam.diem, tam.kq);
    if (tam.diem = = 4 &&strcmp(tam.phai,"nữ")= =0 )
     strcpy(&tam.kq, "đậu");
    fwrite(&tam,size of(tam),1, fr);
    }
    fclose (fp); fclose(fr);
    printf ("\n in file ketqua.txt sau khi xét lại kết qủa ");
    fp = fopen ("c:\\ketqua.txt", "rb");
    while (fread(&tam, size of (KieuHV) , 1, fp) > 0)
     printf("\n %s%s%d%s",tam.ten,tam.phai, tam.diem,tam.kq);
     fclose (fp); getch( );
    }
```

Các hàm xuất nhập ngẫu nhiên

- Khi mở tệp tin để đọc hay ghi, con trỏ chỉ vị luôn luôn ở đầu tập tin (byte 0) nếu mở mode "a" (append)

chuyển con trỏ chỉ vị trí cuối tập tin.

+ Hàm void rewind (FILE*fp) : chuyển con trỏ chỉ vị của tập fp về đầu tập tin.

+ Hàm int fseek (FILE*fp, long số byte, int xp)

fp : là con trỏ tập tin; số byte : là số byte cần di chuyển.

xp " cho biết vị trí xuất phát mà việc dịch chuyển được bắt đầu từ đó.

xp = SEEK - SET hay 0 xuất phát từ đầu tập.

xp = SEEK - CUR hay 1 : xuất phát từ vị trí hiện tại của con trỏ.

xp= SEEK - END HAY 2 : xuất phát từ vị trí cuối tập của con trỏ.

+ Công dụng : hàm di chuyển con trỏ chỉ vị của tập fp từ vị trí xác định bởi xp qua một số byte bằng giá trị tuyệt đối của số byte. Nếu số byte > 0 : chuyển về hướng cuối tập ngược lại chuyển về hướng đầu tập. Nếu thành công trả về trị 0. Nếu có lỗi trả khác 0.

+ Chú ý : không nên dùng fseep trên kiểu văn bản, vì sự chuyển đổi ký tự(mã 10) sẽ làm cho việc định vị thiếu chính xác.

+ Hàm long ftell(FILE*fp) ; : cho biết vị trí hiện tại của con trỏ chỉ vị (byte thứ mấy trên tập fp) nếu không thành công trả về trị -1L.

Ví dụ 18: giả sử chúng ta có tập tin c:\lop.txt chứa danh sách các học viên. Hãy đọc danh sách và sắp xếp giảm dần theo điểm sau đó ghi lại file c:\lop.txt (nối điểm)

```
#include <stdio.h>
#include<conio.h>
#include<string.h>
#define N 100
struct
{ char ten[20] ; int tuoi; float diem ; } KieuHV ;
void main( )
{ KieuHV hv[N] ; t;
FILE*fp ; int i, , n ;
fp = fopen ("c:\\lop.txt ", "rat");
if (fp = =NULL)
  { printf ("không mở được file "); exit(1); }
    n = 0 ; i = 0 ;
```

```
    while (!feof (fp))
       { fread (&hv[i], size of (KieuHV), 1,fp);
        i++; n++ ;
 /* sắp xếp giảm dần theo điểm */
      for (i=0, i <n-1, i++)
      for (j=i+1; j<n, j++)
          if (hv[i].diem <hv[j].diem)
              { t =hv[i] ; hv[i] = hv[j] ; hv[j] = t }
/* ghi lên đĩa */
fseek (fp, 0, SEEK-END);
for ( i=0; i<n ; i++)
fwrite(&hv[i], size of (KieuHV), 1, fp);
}
```

Ví dụ 19: Hiển thị từng ký tự có trong tập tin văn bản ra màn hình.

Phiên bản 1: Dùng biến ký tự

Tên tập tin: typex.cpp

```
#include <stdio.h>

void main(int argc, char *argv[])
{
 FILE *fp;
 char c;

 if (argc <= 1)
   printf("\nCach su dung : \n typex <ten tap tin>");
 else
   if ((fp = fopen(argv[1], "r")) == NULL)
     printf("\nKhong the mo tap tin %s", argv[1]);
   else
   {
```

```
      while ((c = fgetc(fp)) != EOF)
        putc(c, stdout);
    }
  getch();
}
```

Phiên bản 2: Dùng mảng ký tự

Tên tập tin: typex.cpp

```
#include <stdio.h>
#include <string.h>

void main(int argc, char *argv[])
{
  FILE *fp;
  char s[255];

  if (argc <= 1)
    printf("\nCach su dung : \n typex <ten tap tin>");
  else
    if ((fp = fopen(argv[1], "r")) == NULL)
      printf("\nKhong the mo tap tin %s", argv[1]);
    else

      while (fgets(s, 255, fp))
      {
        s[strlen(s) - 1] = 0;
        if (strlen(s))
          puts(s);
      }
  getch();
}
```

Ví dụ 20: Chương trình sau đọc hai tập tin và nối 2 tập tin thành tập thứ 3.

```
#include <stdio.h>
void main()
{
   FILE *fp1, *fp2, *fpout;
   char sf1[50], sf2[50], sfout[50];
   int c;

   printf("\nNhap ten tap tin thu nhat : ");
   scanf("%s", &sf1);
   printf("\nNhap ten tap tin thu hai : ");
   scanf("%s", &sf2);
   printf("\nNhap ten tap tin ket qua : ");
   scanf("%s", &sfout);
   if ((fp1 = fopen(sf1, "r")) == NULL)
     fprintf(stderr, "Khong the mo tap tin %s\n", sf1);
   if ((fp2 = fopen(sf2, "r")) == NULL)
     fprintf(stderr, "Khong the mo tap tin %s\n", sf2);
   if ((fpout = fopen(sfout, "w")) == NULL)
     fprintf(stderr, "Khong the mo tap tin %s\n", sfout);

   while ((c = getc(fp1)) != EOF)
      putc(c, fpout);
   while ((c = getc(fp2)) != EOF)
      putc(c, fpout);
   fclose(fp1);
   fclose(fp2);
   fclose(fpout);

   printf("\nHoan tat! Nhan phim bat ky de ket thuc.");
```

```
   getch();
}
```

Ví dụ 21: Chương trình sẽ mã hóa từng ký tự trong tập tin với một ký tự nhập vào theo phép toán XOR.

```
#include <stdio.h>

void main()
{
  char c, filein[50], fileout[50], key;
  FILE *fpin, *fpout;

  printf("\nCho biet ten tap tin nguon : ");
  gets(filein);
  printf("\nCho biet ten tap tin dich : ");
  gets(fileout);
  printf("\nCho biet khoa : ");
  scanf("%c", &key);
  if ((fpin = fopen(filein, "r")) == NULL)
    printf("Khong tim thay tap tin %s", filein);
  else
    if ((fpout = fopen(fileout, "w+")) == NULL)
    {
      printf("Khong the tao tap tin %s", fileout);
      fclose(fpin);
    }
    else
    {
      do {
        c = fgetc(fpin);
        if (c != EOF)
```

```
            fputc(c ^ key, fpout); //XOR
        } while (c != EOF);
        fclose(fpin);
        fclose(fpout);
        printf("\nĐã mã hóa xong");
      }
   getch();
}
```

2.5. Các kiểu dữ liệu khác

Kiểu tự định nghĩa typedef

C++ cho phép chúng ta định nghĩa các kiểu dữ liệu của riêng mình dựa trên các kiểu dữ liệu đã có. Để có thể làm việc đó chúng ta sẽ sử dụng từ khoá typedef, dạng thức như sau:

typedef kiểu_dữ_liệu_đã_có kiểu_dữ_liệu_mới;

trong đó kiểu_dữ_liệu_đã_có là một kiểu dữ liệu cơ bản hay bất kì một kiểu dữ liệu đã định nghĩa và kiểu_dữ_liệu_mới là tên của kiểu dữ liệu mới. Ví dụ

```
typedef char C;
typedef unsigned int WORD;
typedef char * string_t;
typedef char field [50];
```

Trong trường hợp này chúng ta đã định nghĩa bốn kiểu dữ liệu mới: C, WORD, string_t và field kiểu char, unsigned int, char* kiểu char[50], chúng ta hoàn toàn có thể sử dụng chúng như là các kiểu dữ liệu hợp lệ:

```
achar, anotherchar, *ptchar1;
WORD myword;
string_t ptchar2;
field name;
```

typedef có thể hữu dụng khi bạn muốn định nghĩa một kiểu dữ liệu được dùng lặp đi lặp lại trong chương trình hoặc kiểu dữ liệu bạn muốn dùng có tên quá dài và bạn muốn nó có tên ngắn hơn.

Kiểu liệt kê (enum)

Kiểu dữ liệu liệt kê dùng để tạo ra các kiểu dữ liệu chứa một cái gì đó hơi đặc biệt một chút, không phải kiểu số hay kiểu kí tự hoặc các hằng true và false. Dạng thức của nó như sau:

```
enum model_name {
  value1,
  value2,
  value3,
  .
  .
} object_name;
```

Chẳng hạn, chúng ta có thể tạo ra một kiểu dữ liệu mới có tên color để lưu trữ các màu với phần khai báo như sau:

```
enum colors_t {black, blue, green, cyan, red, purple, yellow, white};
```

Chú ý rằng chúng ta không sử dụng bất kì một kiểu dữ liệu cơ bản nào trong phần khai báo. Chúng ta đã tạo ra một kiểu dữ liệu mới mà không dựa trên bất kì kiểu dữ liệu nào có sẵn: kiểu color_t, những giá trị có thể của kiểu color_t được viết trong cặp ngoặc nhọn {}. Ví dụ, sau khi khai báo kiểu liệt kê, biểu thức sau sẽ là hợp lệ:

```
colors_t mycolor;
mycolor = blue;
if (mycolor == green) mycolor = red;
```

Trên thực tế kiểu dữ liệu liệt kê được dịch là một số nguyên và các giá trị của nó là các hằng số nguyên được chỉ định. Nếu điều này không đươc chỉ định, giá trị nguyên tương đương với phần tử đầu tiên là 0 và các giá trị tiếp theo cứ thế tăng lên 1, Vì vậy, trong kiểu dữ liệu colors_t mà chúng ta định nghĩa ở trên, white tương đương với 0, blue tương đương với 1, green tương đương với 2 và cứ tiếp tục như thế.

Nếu chúng ta chỉ định một giá trị nguyên cho một giá trị nào đó của kiểu dữ liệu liệt kê (trong ví dụ này là phần tử đầu tiên) các giá trị tiếp theo sẽ là các giá trị nguyên tiếp theo, chẳng hạn:

```
enum months_t { january=1, february, march, april,
                may, june, july, august,
                september, october, november, december} y2k;
```

trong trường hợp này, biến y2k có kiểu dữ liệu liệt kê months_t có thể chứa một trong 12 giá trị từ january đến december và tương đương với các giá trị nguyên từ 1 đến 12, không phải 0 đến 11 vì chúng ta đã đặt january bằng 1.

Ví dụ 22: Chương trình sau kết hợp giữa kiểu liệt kê và kiểu cấu trúc để tính lương cho một tuần làm việc. Chủ nhật được tính giờ gấp đôi, thứ năm được tính 1,25 giờ cho một giờ làm. Giá tiền trả cho một giờ lương là 1,1 USD.

```
#include"stdio.h"
enum luong_tuan{
Thu_Hai,Thu_Ba,Thu_tu,Thu_Nam,Thu_Sau,Thu_Bay,Chu_Nhat};
char *thu[]={"Thu Hai","Thu Ba","Thu Tu","Thu Nam","Thu Sau","Thu Bay","Chu Nhat"};
void main()
   {
        enum luong_tuan ngay;
        char ten[8];
        float luong=0;
        int gio;
        clrscr();
        printf("\nCho biet ten : ");
        scanf("%s",&ten);
        for (ngay=Thu_Hai; ngay <=Chu_Nhat;ngay++)
              {
                   printf("\n- Gio lam viec trong ngay: %s la= ",thu[(int)ngay]);
                   scanf("%d",&gio);
                   switch (ngay)
                        {
                             case Thu_Nam: luong+=1.25*gio;
                              break;
                             case Chu_Nhat: luong+=2*gio;
                               break;
```

```
                                    default             : luong+=1.1*gio;
                                     break;
                                }
                        }
                printf("\n Ong ( Ba ) : %s ",ten);
                printf("\n-Se nhan duoc tien luong trong tuan la = ");
                printf("%d dollars va %0.2f cents",(int)luong,luong-(int)luong);
                getch();
        }
```

2.6. Bài tập

Bài tập 1: Hãy viết chương trình nhập vào một dãy các số nguyên. Lưu dãy số nguyên này vào file.

Bài tập 2: Tìm kiếm một số nguyên trong dãy các số nguyên có trong file ở bài tập 1.

Bài tập 3: Viết chương trình giải hệ phương trình dùng phương pháp Gauss-Jordan.

Bài tập 4: Viết chương trình quản lý sách trong thư viện.

Bài tập 5: Viết chương trình đọc hai ma trận vuông từ hai tập tin. Thực hiện các phép toán:

- Nhân hai ma trận.
- Cộng hai ma trận.
- Tính đa thức ma trận.
- Trừ hai ma trận
- Tính AT

Bài tập 6: Vẽ sơ đồ các kiểu dữ liệu đã học trong bài.

Bài tập 7: Viết chương trình với hai tập tin, một tập tin dùng để mã hóa, một tập tin chứa dữ liệu. Thực hiện, đọc tập tin dữ liệu và đọc tập tin mã hóa, sau đó dùng phép toán XOR để mã hóa từng ký tự đọc được lưu lại vào tập tin thứ 3.

CHƯƠNG 3: MẢNG, DANH SÁCH VÀ CÁC KIỂU DỮ LIỆU TRỪU TƯỢNG

Mã bài: MH28.3

Mục tiêu:

- Trình bày được khái niệm, cấu trúc lưu trữ của dữ liệu kiểu mảng, kiểu danh sách;
- Sử dụng được một số phép toán xử lý trên các phần tử của danh sách liên kết;
- Sử dụng cấu trúc, các phép xử lý, khả năng áp dụng của ngăn xếp, hàng đợi;
- Viết được một số giải thuật xử lý các yêu cầu cụ thể trên các kiểu dữ liệu trên;
- Cài đặt được một số thao tác xử lý danh sách liên kết, ngăn xếp, hàng đợi trên ngôn ngữ C, Pascal;
- Nghiêm túc, tỉ mỉ, sáng tạo trong việc học và vận dụng vào làm bài tập. Chủ động kết hợp các ngôn ngữ lập trình để cài đặt thuật toán.

3.1 Mảng

Kiểu dữ liệu mảng là một tập hợp có thứ tự chứa các phần tử có cùng kiểu dữ liệu được lưu trữ liên tiếp nhau trong bộ nhớ. Mảng có thể một chiều hay nhiều chiều.

Mảng một chiều được khai báo như sau:

<tên kiểu dữ liệu> tênbiến[<kích thước>]

Ví dụ 1: Để khai báo một biến a là một mảng nguyên một chiều có 100 phần tử, chúng ta phải khai báo như sau:

```
int  a[100];
```

Để truy xuất đến các phần tử,chúng ta phải chỉ định tên và chỉ số phần tử cần truy xuất. Chẳng hạn:

a[0] - Truy xuất đến phần tử đầu tiên.

a[11] – Truy xuất đến phần tử thứ 12 (có chỉ số 11)

a[99] – Truy xuất đến phần tử thứ 100 (có chỉ số 99), là phần tử cuối cùng ở mảng trên.

Chúng ta cũng có thể vừa khai báo vừa gán giá trị cho một mảng như sau:

```
int a [5] = { 16, 2, 77, 40, 12071 };
```

Tương tự, chúng ta có thể khai báo một mảng 2 hay nhiều chiều theo cú pháp sau:

<tên kiểu dữ liệu> tênbiến[<kích thước1>][<kích thước 2>][…];

Chẳng hạn, chúng ta khai báo khai báo mảng nguyên 2 chiều như sau:

```
int  a[100][20];
```

Ví dụ 2: Nhập một mảng 1 chiều có tối đa 100 phần tử, số phần tử được nhập từ bàn phím. Hãy in tổng các phần tử không chia hết cho 2 và các phần tử chia hết cho 2.

```
#include <conio.h>
#include <iostream.h>
void main()
{
 int a[100]; //toi da 100 phan tu
 int n,i;
 long tongle,tongchan;
 clrscr();
 cout<<"Nhap vao so phan tu:";cin>>n;
 for (i=0;i<=n-1;i++)
 {
  cout<<"Nhap gia tri phan tu thu "<<i<<": ";
  cin>>a[i];
 }
 tongle=0;tongchan=0;
 for (i=0;i<=n-1;i++)
  if (a[i]%2==0)
   tongchan+=a[i];
  else
   tongle+=a[i];
 cout<<"Tong cac phan tu khong chia het cho 2:"<<tongle<<endl;
 cout<<"Tong cac phan tu chia chet cho 2:"<<tongchan;
 getch();
}
```

Ví dụ 3: Nhập vào hai ma trận $A_{m\times n}, B_{m\times n}$ với m,n được nhập từ bàn phím. Sau đó in ra ma trận C là ma trận có được khi cộng ma trận A cho B.

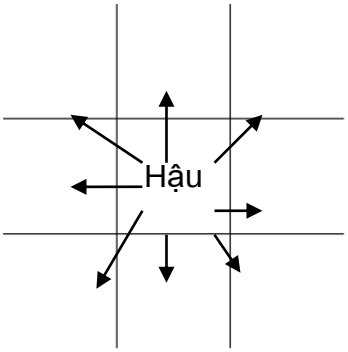
```
#include <conio.h>
#include <iostream.h>
```

```
void main()
{
 int a[100][100],b[100][100];
 int c[100][100];
 int n,m,i,j;
 clrscr();
 cout<<"Nhap m=";cin>>m;
 cout<<"Nhap n=";cin>>n;
 cout<<"Nhap cac phan tu cho ma tran A"<<endl;
 for (i=0;i<m;i++)
  for (j=0;j<n;j++)
   {
     cout<<"a["<<i<<"]["<<j<<"]=";
     cin>>a[i][j];
   }
 cout<<"Nhap cac phan tu cho ma tran B"<<endl;
 for (i=0;i<m;i++)
  for (j=0;j<n;j++)
   {
     cout<<"b["<<i<<"]["<<j<<"]=";
     cin>>b[i][j];
   }
  //tinh ma tran C
  for (i=0;i<m;i++)
   for (j=0;j<n;j++)
    c[i][j]=a[i][j]+b[i][j];
  cout<<"cac phan tu trong ma tran C:"<<endl;
  for (i=0;i<m;i++)
   for(j=0;j<n;j++)
    cout<<"c["<<i<<"]["<<j<<"]="<<c[i][j]<<endl;
```

```
  getch();
}
```

Mảng được xử lý rất uyển chuyển nhờ phép toán truy cập trực tiếp dữ liệu thông qua chỉ số phần tử, do đó, mảng thường được dùng để lưu trữ các dữ liệu trong những chương trình mà khả năng lưu trữ dữ liệu không lớn. Đồng thời, mảng vẫn là cấu trúc dữ liệu hay được dùng trong các giải pháp xử lý đệ quy-quay lui, hay xử lý tham lam, ... Chúng ta tiến hành nghiên cứu việc ứng dụng mảng trong một số phương pháp.

Ví dụ 3: Bài toán tám quân hậu: Tìm cách đặt các quân hậu lên bàn cờ vua sao cho không quân nào có thể ăn được quân nào. Quy luật khống chế ô cờ của quân hậu được thể hiện theo hướng mũi tên sau:



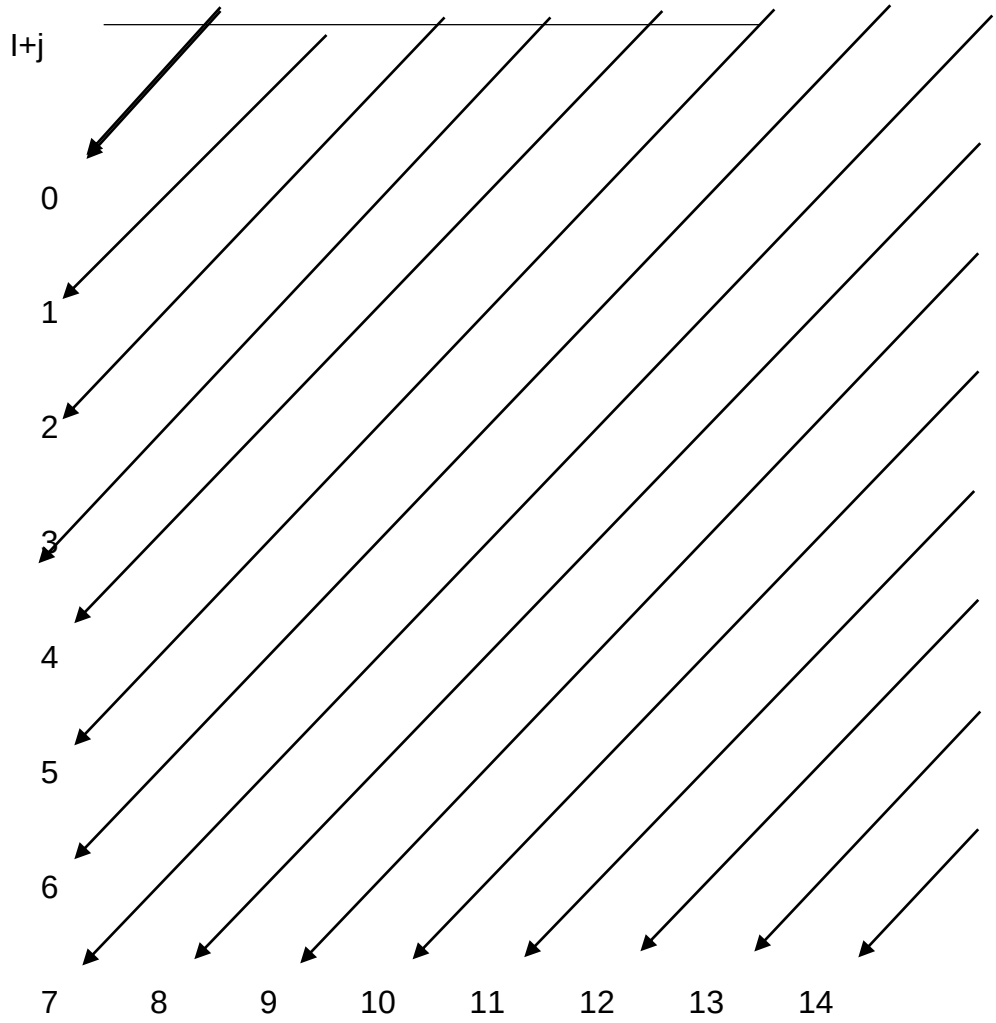
Thuật toán xây dựng dựa trên việc đặt quân hậu lần lượt trên từng dòng. Chúng ta phải tìm cột thích hợp để đặt quân hậu lên dòng thứ i, tại cột j nào?

- Giá trị cột đã bị quân hậu khống chế rất dễ dàng xác định bằng cách xem có quân hậu nào nằm trên cột đó hay không?

- Các đường chéo xuôi được xác định dễ dàng vì lấy chỉ số cột trừ chỉ số dòng trên một đường chéo luôn luôn là một hằng số.

- Cac đường chéo ngược được xác định dễ dàng vì lấy chỉ số dòng cộng chỉ số cột trên một đường chéo luôn luôn là một hằng số.

Bảng chỉ số i+j của đường chéo xuôi.



Bài toán này chúng ta xử lý bằng cách dùng một mảng để lưu lời giải (mảng loigiai). Hai mảng dùng để lưu đường chéo của bàn cờ (cheoxuoitrong, cheonguoctrong), một mảng lưu trữ các cột trống

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <iostream.h>

#define KICHTHUOC 8                    // Kich thuoc cua ban co
#define SODUONGCHEO (2*KICHTHUOC-1)    // So duong cheo cua ban co
#define SOGIA (KICHTHUOC-1)            // so gia
```

```
#define TRUE 1
#define FALSE 0
// prototypes
void hoanghau(int);
void inloigiai(int loigiai[]);

int cottrong[KICHTHUOC];
 // mang cac cot co the dat hoang hau
int cheoxuoitrong[SODUONGCHEO];    // mang cac duong cheo xuoi co the dat hhau
int cheonguoctrong[SODUONGCHEO];   // mang cac duong cheo nguoc co the dat hhau

int loigiai[KICHTHUOC];
/* mang loi giai cho biet cot dat cac hoang hau tren ban co. Vi du cac phan tu cua mang la: 7  3  0  2  5  1  6  4          cho biet hoang hau 0 dat o cot 7, hoanghau 1 dat o cot 3 , ..., hoanghau 7 o cot 4      */
int SoLoiGiai = 0;

void main(void)
{
  int i;

  /* Khoi dong tat ca cac cot duong cheo xuoi, duong cheo nguoc deu co the dat hoang hau */
  for(i = 0; i < KICHTHUOC; i++)
    cottrong[i] = TRUE;
  for(i = 0; i < SODUONGCHEO; i++)
  {

    cheoxuoitrong[i] = TRUE;
```

```
    cheonguoctrong[i] = TRUE;
  }

  // Goi ham de qui de bat dau dat HoangHau0 (hoang hau o hang 0)
  hoanghau(0);
}

// Ham hoanghau giup dat hoang hau i (i tu 0 den KICHTHUOC-1) tren hang i
void hoanghau(int i)
{
  int j;
  for(j = 0; j < KICHTHUOC; j++)
    if(cottrong[j] && cheoxuoitrong[i-j+SOGIA] && cheonguoctrong[i+j])
    {
      // Dat hoang hau vao o (i, j) tren ban co
      loigiai[i] = j;
      cottrong[j] = FALSE;
      cheoxuoitrong[i-j+SOGIA] = FALSE;
      cheonguoctrong[i+j] = FALSE;

      if(i == KICHTHUOC-1)
       // Dkien dung, dat duoc con hoang hau cuoi
        inloigiai(loigiai);
      else              // Buoc de qui, goi dat hoang hau i+1
        hoanghau(i + 1);

      // lan nguoc
      cottrong[j] = TRUE;
      cheoxuoitrong[i-j+SOGIA] = TRUE;
      cheonguoctrong[i+j] = TRUE;
```

```
        }
    }
    void inloigiai(int loigiai[])
    {
      int  j;
      SoLoiGiai++;
      cout<<"Loi giai thu "<<SoLoiGiai;

      for (j=0;j<n;j++)
      cout<<"Hoang hau "<< j << "dat co cot " << loigiai[j] <<" dong " <<j <<endl;

      clrscr();
    }
```

Tương tự, bài toán mã đi tuần sau cũng sử dụng cấu trúc mảng để giải quyết một cách dễ dàng.

Ví dụ 4: Bài toán mã đi tuần: Đặt 1 quân mã vào bàn cờ vua, hãy tìm đường đi để quân mã đi hết bàn cờ mà không đi lại bất cứ ô nào.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define KICHTHUOC 5  // Kich thuoc cua ban co
void nuocdi(int, int, int);
void innuocdi();
// To chuc ban co la mang hai chieu
int BanCo[KICHTHUOC][KICHTHUOC];
// 8 cach di cua con ma
int a[8] = {2,  2,  1, -1, -2, -2, -1,  1};
int b[8] = {1, -1, -2, -2, -1,  1,  2,  2};
int SoLoiGiai = 0;
int main(void)
{
```

```
    int i, j, m, n;
    clrscr();
    for(i = 0; i < KICHTHUOC; i++)
      for(j = 0; j < KICHTHUOC; j++)
      {
         // Khoi dong tat ca cac o tren ban co deu chua di
         for(m = 0; m < KICHTHUOC; m++)
           for(n = 0; n < KICHTHUOC; n++)
             BanCo[m][n] = 0;
         // Chon nuoc di dau tien va goi ham de qui de di nuoc thu hai
         BanCo[i][j] = 1;
         nuocdi(2, i, j);
      }
    return 0;
  }
  // Ham NuocDi giup di nuoc thu n xuat phat tu o(x, y)
  void nuocdi(int n, int x, int y)
  {
    int i;
    char c;
    for(i = 0; i < 8; i++)
    {
      if(x+a[i] >= 0 && x+a[i] < KICHTHUOC && y+b[i] >= 0 && y+b[i] <
KICHTHUOC &&BanCo[x+a[i]][y+b[i]] == 0)
      {
         // Di nuoc thu n
         BanCo[x+a[i]][y+b[i]] = n;

         if(n == KICHTHUOC*KICHTHUOC)  // Dkien dung, di duoc nuoc cuoi
         {
```

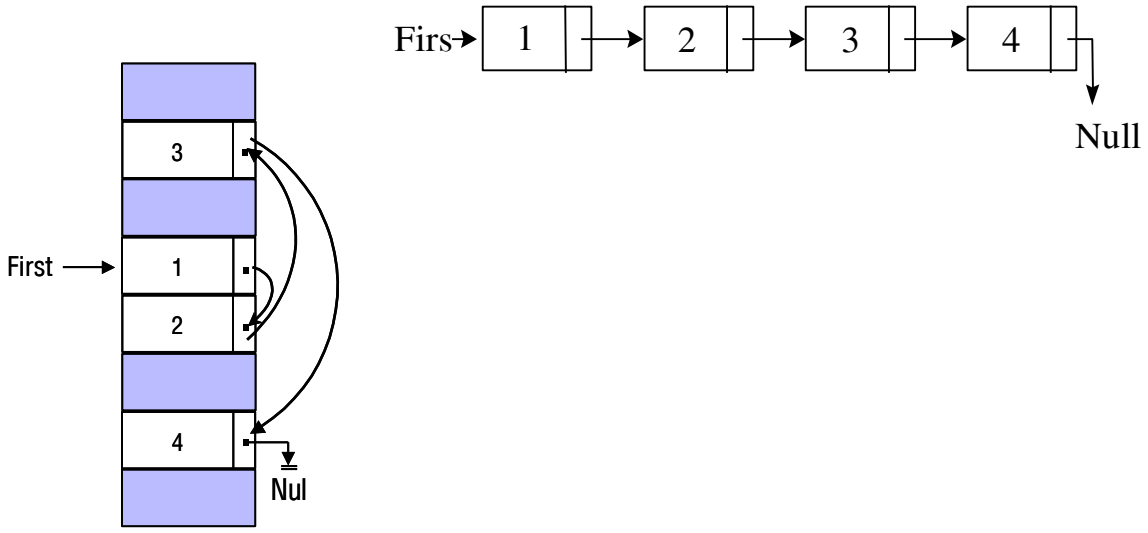
```
            innuocdi();
                  }
            else                    // Buoc de qui, goi di nuoc n+1
              nuocdi(n+1, x+a[i], y+b[i]);

            // lan nguoc
            BanCo[x+a[i]][y+b[i]] = 0;
        }
      }
}
void inloigiai()
{
  int i,j;
  cout<<"Loi giai";
  for (i=0;i<8;i++)
   {
     for (j=0;j<8;j++)
      printf("%3d",BanCo[i][j]);
}
```

3.2. Danh sách liên kết

I. KHÁI NIỆM:

Cấu trúc danh sách liên kết là cấu trúc động, việc cấp phát nút và giải phóng nút trên danh sách xảy ra khi chương trình đang chạy. Ta thường cấp phát nút cho danh sách liên kết bằng biến động. Danh sách liên kết có nhiều loại như danh sách liên kết đơn, danh sách liên kết kép, danh sách liên kết đa và danh sách liên kết vòng. Trong chương này ra sẽ khảo sát một số loại danh sách liên kết như đơn, kép và vòng.

Các phần tử trong danh sách liên kết sẽ được cấp phát vùng nhớ trong quá trình thực thi chương trình, do đó chúng có thể nằm rải rác ở nhiều nơi khác nhau trong bộ nhớ (không liên tục).



Hình 4.1 Minh họa danh sách liên kết trong bộ nhớ

Các phần tử trong danh sách được kết nối với nhau theo chùm liên kết như hình trên:

- First là con trỏ chỉ đến phần tử đầu của danh sách liên kết

- Phần tử cuối của danh sách liên kết với vùng liên kết có giá trị NULL

- Mỗi nút của danh sách có trường **info** chứa nội dung của nút và trường **next** là con trỏ chỉ đến nút kế tiếp trong danh sách.

* ***Lưu ý***:

- Cấu trúc danh sách liên kết là cấu trúc động, các nút được cấp phát hoặc bị giải phóng khi chương trình đang chạy.

- Danh sách liên kết rất thích hợp khi thực hiện các phép toán trên danh sách thường bị biến động. Trong trường hợp xóa hay thêm phần tử trong danh sách liên kết thì ta không dời các phần tử đi như trong danh sách tuyến tính (mảng) mà chỉ việc hiệu chỉnh lại trường next tại các nút đang thao tác. Thời gian thực hiện các phép toán thêm vào và loại bỏ không phụ thuộc vào số phần tử của danh sách liên kết.

- Tuy nhiên, danh sách liên kết cũng có các điểm hạn chế sau:

+ Vì mỗi nút của danh sách liên kết phải chứa thêm trường next nên danh sách liên kết phải tốn thêm bộ nhớ.

+ Tìm kiếm trên danh sách liên kết không nhanh vì ta chỉ được truy xuất tuần tự từ đầu danh sách.

▪ Khai báo : Một phần tử của danh sách liên kết ít nhất phải có hai thành phần : nội dung của phần tử (info) và thành phần next để liên kết phần tử này với phần tử

khác.

Giả sử ta khai báo kiểu NODEPTR là kiểu con trỏ chỉ đến nút trong 1 danh sách liên kết, mỗi phần tử có 2 thành phần : info (số nguyên) và next.

```
struct node
{  int info ;
   struct node *next ;
};
typedef struct node *NODEPTR;
```

- Để khai báo biến First quản lý danh sách liên kết ta viết như sau:

```
NODEPTR First;
```

- Khởi tạo danh sách liên kết : First = NULL;

- Ghi chú :

· Thành phần chứa nội dung có thể gồm nhiều vùng với các kiểu dữ liệu khác nhau.

Ví dụ: Khai báo biến First để quản lý một danh sách sinh viên với cấu trúc dữ liệu là danh sách liên kết đơn, mỗi sinh viên gồm có 2 thành phần là: mssv (số nguyên) và họ tên.

```
struct sinhvien
  int mssv;
  char hoten[30];
};
struct node
{  sinhvien sv ;
   struct node *next ;
};
typedef struct node *NODEPTR;
NODEPTR First;
```

· Thành phần liên kết cũng có thể nhiều hơn một nếu là danh sách đa liên kết hoặc danh sách liên kết kép.

· First là con trỏ trỏ đến phần tử đầu tiên của danh sách liên kết, nó có thể là kiểu con trỏ (như khai báo trên), và cũng có thể là một struct có hai thành phần: First trỏ

đến phần tử đầu tiên của danh sách liên kết, và Last trỏ đến phần tử cuối của danh sách liên kết.

```
struct Linked_List;
{  First NODEPTR;
   Last NODEPTR;
};
```

II. CÁC PHÉP TOÁN TRÊN DANH SÁCH LIÊN KẾT:

Để đơn giản, các phép toán sau đây sẽ được thao tác trên danh sách các số nguyên với khai báo như sau:

```
struct node
{  int info ;
   struct node *next ;
};
typedef struct node *NODEPTR;
```

- Để khai báo biến First quản lý danh sách liên kết ta viết như sau:

```
NODEPTR First;
```

II.1. ***Tạo danh sách*:**

a. Khởi tạo danh sách (Initialize): dùng để khởi động một danh sách liên kết, cho chương trình hiểu là hiện tại danh sách liên kết chưa có phần tử.

```
void Initialize(NODEPTR &First)
{
  First = NULL;
}
```

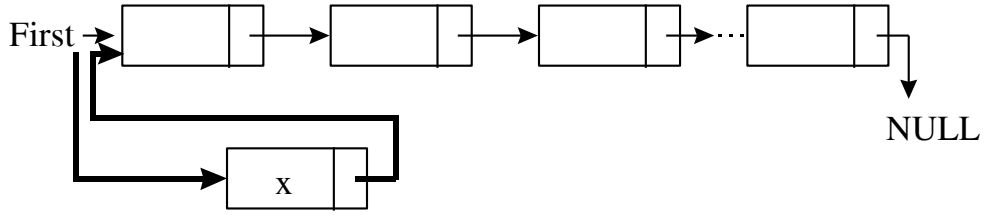
b. Cấp phát vùng nhớ (New_Node): cấp phát một nút cho danh sách liên kết. Hàm New_Node này trả về địa chỉ của nút vừa cấp phát.

Trong chương trình có sử dụng hàm malloc (trong <alloc.h>), hàm này cấp phát một khối nhớ tính theo byte từ bộ nhớ heap. Nếu cấp phát thành công, hàm malloc trả về địa chỉ của vùng nhớ vừa cấp phát, ngược lại nó sẽ trả về NULL.

```
NODEPTR New_node(void)
{
  NODEPTR p;
```

```
    p = (NODEPTR)malloc(sizeof(struct node));
    return (p);
}
```

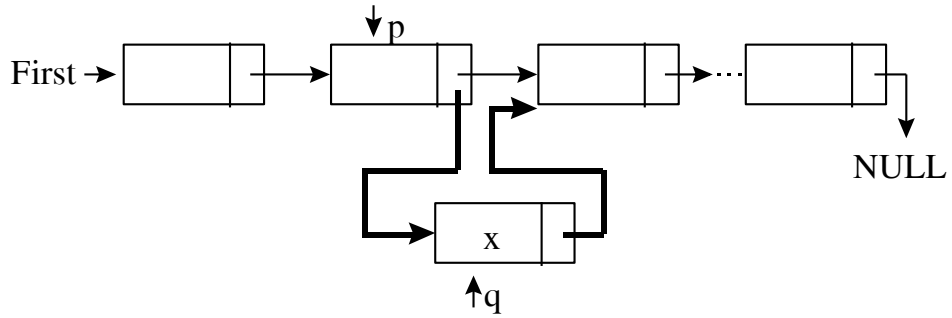
c. <u>Thêm vào đầu danh sách</u> (Insert_first): thêm một nút có nội dung x vào đầu danh sách liên kết.



Hình 4.2 Thêm nút có nội dung x vào đầu danh sách liên kết

```
void Insert_first(NODEPTR &First, int x)
{
  NODEPTR p;
  p = New_node();
  p->info = x;
  p->next = First;
  First = p;
}
```

d. <u>Thêm nút mới vào sau nút có địa chỉ p</u> (Insert_after): thêm một nút có nội dung x vào sau nút có địa chỉ p trong danh sách liên kết First.



Hình 4.3 Thêm nút có nội dung x vào sau nút có địa chỉ p

```
void Insert_after(NODEPTR p, int x)
{
  NODEPTR q;
      if(p == NULL)
```

```
            printf("khong them phan tu vao danh sach duoc");
    else
    {
      q = New_node();
      q->info = x;
      q->next = p->next;
      p->next = q;
    }
}
```

II.2. Tìm kiếm (Search_info):

Tìm nút đầu tiên trong danh sách có info bằng với x. Do đây là danh sách liên kết nên ta phải tìm từ đầu danh sách.

Hàm Search_info nếu tìm thấy x trong danh sách thì trả về địa chỉ của nút có trị bằng x trong danh sách, nếu không có thì trả về trị NULL.

```
NODEPTR Search_info(NODEPTR First, int x)
{
  NODEPTR p;
  p = First;
  while(p != NULL && p->info != x  )
          p = p->next;
  return(p);
}
```

II.3. Cập nhật danh sách:

a. Giải phóng vùng nhớ (free): Hàm này dùng để hủy nút đã cấp phát, và trả vùng nhớ về lại cho memory heap.

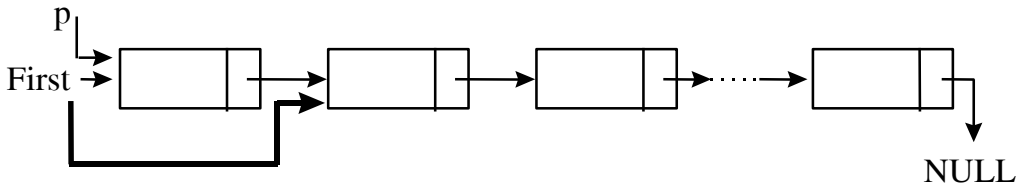
free(p) ; với p là biến con trỏ

b. Kiểm tra danh sách liên kết rỗng hay không (Empty): hàm Empty trả về TRUE nếu danh sách liên kết rỗng, và ngược lại.

```
int Empty(NODEPTR First)
{
```

```
  return(First == NULL ? TRUE : FALSE);
}
```

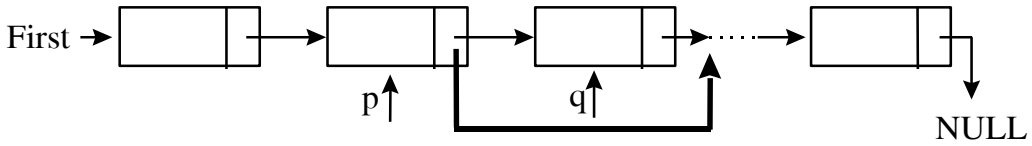
c. <u>Xóa phần tử đầu của danh sách (Delete_First):</u> muốn xóa 1 phần tử khỏi danh sách liên kết thì ta phải kiểm tra xem danh sách có rỗng hay không. Nếu danh sách có phần tử thì mới xóa được.



Hình 4.4 Xóa nút đầu tiên trong danh sách liên kết

```
void Delete_First (NODEPTR &First)
{ NODEPTR p;
  if (Empty(First))
    printf("Danh sach rong nen khong the xoa");
  else
  {
    p = First;    // nut can xoa la nut dau
    First = p->next;
    free(p);
  }
}
```

d. <u>Xóa phần tử đứng sau nút có địa chỉ p (Delete_after)</u>:



Hình 4.5 Xóa nút sau nút có địa chỉ p

```
void Delete_after(NODEPTR p)
{   NODEPTR q;
 // nếu p là NULL hoặc sau p không có nút
  if((p == NULL) || (p->next == NULL))
    printf("Khong xoa duoc");
```

```
  else
  {
    q = p->next;  // q chi nut can xoa
   p->next = q->next;
   free(q);
  }
}
```

e. <u>Xóa phần tử theo nội dung (Delete_info):</u>

- Tìm địa chỉ của phần tử có nội dung là x trong danh sách liên kết.
- Loại bỏ phần tử này theo địa chỉ.

```
void Delete_info(NODEPTR &First,int x)
{ NODEPTR q,p;
  p= search_info(First,x);
  if (p==NULL)
    printf("Khong co gia tri %d trong danh sach", x);
  else
  {
    if (p==First)
       Delete_first(First);
    else
    {
       q=First;
       while (q->next != p)
          q=q->next;
       Delete_after(q);
    }
    printf("Da xoa phan tu co gia tri %d trong danh sach",x);
  }
}
```

<u>Lưu ý</u> : Giải thuật này chỉ loại bỏ phần tử đầu tiên trong danh sách có giá trị info

= x.

f. <u>Xóa toàn bộ danh sách</u> (Clearlist): ta có thể sử dụng lệnh First = NULL để xóa toàn bộ danh sách, nhưng trong bộ nhớ, các vùng nhớ đã cấp phát cho các nút không giải phóng về lại cho memory heap, nên sẽ lãng phí vùng nhớ. Do đó, ta sử dụng giải thuật sau:

```
void Clearlist(NODEPTR &First)
{
  NODEPTR p;
 while(First != NULL)
  {
    p = First;
    First = First->next;
    free(p);
  }
}
```

II.4. <u>Duyệt danh sách</u>:

Thông thường ta hay duyệt danh sách liên kết để thực hiện một công việc gì đó, như liệt kê dữ liệu trong danh sách hay đếm số nút trong danh sách...

```
void Traverse(NODEPTR First)
{ NODEPTR p;
  int stt = 0;
  p = First;
  if(p == NULL)
    printf("\n   (Khong co phan tu trong danh sach)");
  while(p != NULL)
  {
    printf("\n   %5d%8d", stt++, p->info);
    p = p->next;
  }
}
```

II.5. Sắp xếp (Selection_Sort): sắp xếp danh sách liên kết theo thứ tự info tăng dần.

- Nội dung: Ta so sánh tất cả các phần tử của danh sách để chọn ra một phần tử nhỏ nhất đưa về đầu danh sách; sau đó, tiếp tục chọn phần tử nhỏ nhất trong các phần tử còn lại để đưa về phần tử thứ hai trong danh sách. Quá trình này lặp lại cho đến khi chọn ra được phần tử nhỏ thứ (n-1).

- Giải thuật:

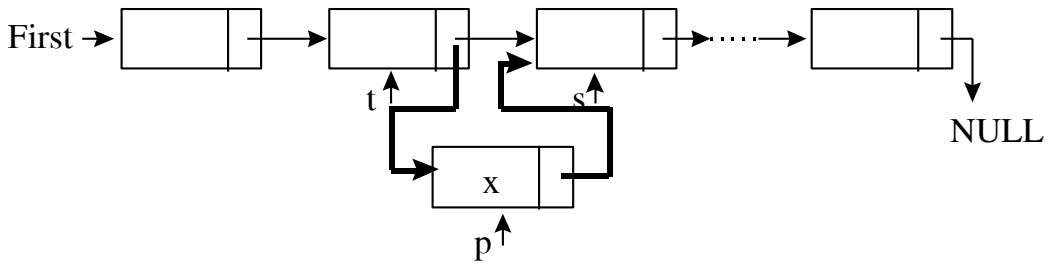
```
void Selection_Sort(NODEPTR &First)
{ NODEPTR p, q, pmin;
      int min;
      for(p = First; p->next != NULL; p = p->next)
      {       min = p->info;
              pmin = p;
              for(q = p->next; q != NULL; q = q->next)
                      if(min > q->info)
                      {
                              min = q->info;
                              pmin = q;
                      }
              // hoan doi truong info cua hai nut p va pmin
              pmin->info = p->info;
              p->info = min;
      }
}
```

III. CÁC PHÉP TOÁN TRÊN DANH SÁCH LIÊN KẾT CÓ THỨ TỰ:

Danh sách liên kết có thứ tự là một danh sách liên kết đã được sắp xếp theo một thứ tự nhất định (tăng hay giảm) trên một thành phần nào đó của nội dung. Ở đây, ta giả sử danh sách liên kết First có thứ tự tăng theo thành phần info.

III.1. Phép thêm vào :

Thêm vào danh sách liên kết có thứ tự một phần tử có nội dung là x sao cho sau khi thêm vào vẫn đảm bảo tính có thứ tự của danh sách.



Hình 4.6 Thêm nút có nội dung x vào danh sách liên kết có thứ tự

* Giải thuật:

- Tạo phần tử mới, gán giá trị x cho nó

```
New_node  (p) ;
p->info = x ;
```

- Tìm vị trí thích hợp để đưa phần tử mới vào, nghĩa là tìm hai vị trí t và s sao cho: t^.info <= x <= s^.info

```
for(s = First; s != NULL && s->info < x; t=s, s = s->next);
```

- Gán liên kết thích hợp sao cho p nằm giữa hai phần tử có địa chỉ t và s:

```
if(s == First)  // them nut vao dau danh sach lien ket
{
  p->next = First;
  First = p;
}
else          // them nut p vao truoc nut s
{
  p->next= s;
  t->next= p;
}
```

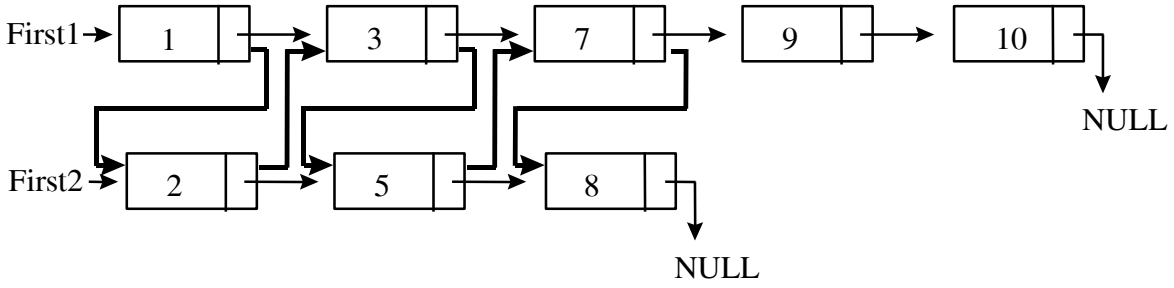
*** Chương trình:**

```
void Insert_Order(NODEPTR &First, int x)
{
  NODEPTR p, t, s;  // q la nut truoc, p la nut sau
  p=New_node();
  p->info=x;
```

```
 for(s = First; s != NULL && s->info < x ; t=s, s = s->next);
if(s == First)  // them nut vao dau danh sach lien ket
  {
    p->next = First;
    First = p;
  }
  else          // them nut p vao truoc nut s
  {
    p->next= s;
    t->next= p;
  }
}
```

III.2. <u>Phép trộn</u> :

Cho hai danh sách liên kết First1, First2 đã có thứ tự. Hãy trộn hai danh sách này lại thành một danh sách liên kết mới First3 sao cho nó cũng có thứ tự.



Hình 4.7 Trộn hai danh sách liên kết có thứ tự

a. <u>Giải thuật</u>: Gọi p1, p2, p3 là 3 biến con trỏ để duyệt 3 danh sách First1, First2, First3

- Tạo giả nút đầu tiên trong danh sách liên kết First3 để hình thành một phần tử cho p3 chỉ đến.

- Duyệt First1 và First2:

Nếu p1->info < p2->info :

Đưa phần tử p1 vào sau phần tử p3

Cho p1 chỉ đến phần tử kế trong danh sách First1

Nếu p2->info < p1->info :

Đưa phần tử p2 vào sau phần tử p3

Cho p2 chỉ đến phần tử kế trong danh sách First2

Quá trình duyệt sẽ dừng lại khi 1 trong 2 danh sách đã duyệt xong

- Đưa nốt phần còn lại của danh sách chưa duyệt xong vào danh sách liên kết First3.

- Xóa phần tử giả đầu danh sách First3 đã tạo ở trên.

```
NODEPTR Merge(NODEPTR First1, NODEPTR First2)
{ NODEPTR p1, p2, p3;
  First3=New_node();
  p1=First1; p2 = First2; p3=First3;
  while (p1 !=NULL && p2 !=NULL)
    if (p1->info < p2->info)
    { p3->next = p1;
      p3=p1;
      p1=p1->next ;
    }
    else
    { p3->next = p2;
      p3=p2;
      p2=p2->next ;
    }
    if (p1==NULL)
         p3->next=p2;
    else
         p3->next=p1;
    p3 = First3;
    First3=p3->next;
    free(p3);
    return First3;
}
```

Ví dụ:

Viết chương trình tạo một menu để quản lý danh sách sinh viên gồm các công việc sau:

1. Tạo danh sách sinh viên: Quá trình nhập danh sách sẽ dừng lại khi ta nhập mã số là 0.
2. Thêm sinh viên vào danh sách: Thêm 1 sinh viên vào danh sách, vị trí sinh viên thêm vào do ta chọn
3. Xem danh sách sinh viên: Liệt kê danh sách sinh viên trên màn hình
4. Hiệu chỉnh sinh viên: nhập vào vị trí sinh viên cần hiệu chỉnh, sau đó chương trình cho phép ta hiệu chỉnh lại mã số, họ, tên của sinh viên.
5. Xóa sinh viên trong danh sách: xóa sinh viên theo vị trí.
6. Tìm kiếm sinh viên theo mã số: nhập vào mã số sinh viên, sau đó in ra vị trí của sinh viên trong danh sách.
7. Sắp xếp danh sách sinh viên theo mã số tăng dần
8. Thêm sinh viên vào danh sách đã có thứ tự tăng dần theo mã số sao cho sau khi thêm thì danh sách vẫn còn tăng dần theo mã số.
9. Xóa toàn bộ danh sách sinh viên.

   Biết rằng:

   - Mỗi sinh viên gồm các thông tin: mã số (int), họ, tên

   - Danh sách sinh viên được tổ chức theo danh sách liên kết đơn.

Chương trình:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include <ctype.h>
#define TRUE 1
#define FALSE 0
struct sinhvien
{
  int mssv;
```

```
  char ho[30];
  char ten[10];
};
struct node
{
  sinhvien sv;
  struct node *next;
};
typedef node *NODEPTR;
 NODEPTR First;
  sinhvien sv;
  NODEPTR p;
// Phep toan New_node: cap phat mot nut cho danh sach lien ket
NODEPTR New_node(void)
{
  NODEPTR p;
  p = (NODEPTR)malloc(sizeof(struct node));
  return(p);
}
/* Tac vu nodepointer: xac dinh con tro cua nut i trong danh sach lien ket
  (i = 2, ...) */
NODEPTR nodepointer(NODEPTR First, int i)
{
     NODEPTR p;
     int vitri=1;
     p = First;
     while(p != NULL && vitri < i)
     {
            p = p->next;
            vitri++;
```

```
    }
    return(p);
}
// Tac vu position: xac dinh vi tri cua nut p trong danh sach lien ket
int position(NODEPTR First, NODEPTR p)
{
    int vitri;
    NODEPTR q;
    q = First;
    vitri = 1;
    while(q != NULL && q != p)
    {
        q = q->next;
        vitri++;
    }
  if(q == NULL)
    return(-1);
  return(vitri);
}
// Phep toan initialize: khoi dong danh sach lien ket
void initialize(NODEPTR &First)
{
  First = NULL;
}
// Tac vu Empty: kiem tra danh sach lien ket co bi rong khong
int Empty(NODEPTR First)
{
  return(First == NULL ? TRUE : FALSE);
}
// Phep toan Insert_first: them nut moi vao dau danh sach lien ket
```

```
void Insert_first(NODEPTR &First, sinhvien x)
{
  NODEPTR p;
  p = New_node();
  p->sv = x;
  p->next = First;
  First = p;
}
// Phep toan Insert_after: them nut moi sau nut co dia chi p
void Insert_after(NODEPTR p, sinhvien x)
{
  NODEPTR q;
  if(p == NULL)
            printf("khong them sinh vien vao danh sach duoc");
  else
  {
    q = New_node();
    q->sv = x;
    q->next = p->next;
    p->next = q;
  }
}
// Phep toan Delete_first: xoa nut o dau danh sach lien ket
void Delete_first(NODEPTR &First)
{
  NODEPTR p;
  if(Empty(First))
    printf("Khong co sinh vien trong danh sach");
  else
  {
```

```
    p = First;    // nut can xoa la nut dau
    First = p->next;
    free(p);
  }
}
// Tac vu Delete_after: xoa nut sau nut p
void Delete_after(NODEPTR p)
{
  NODEPTR q;
  // neu p la NULL hoac p chi nut cuoi
  if((p == NULL) || (p->next == NULL))
    printf("khong xoa sinh vien nay duoc");
  else
  {
    q = p->next;  // q chi nut can xoa
    p->next = q->next;
    free(q);
  }
}
/* Phep toan Insert_Order: Phep toan nay chi su dung khi them nut vao danh sach lien ket da co thu tu */
void Insert_Order(NODEPTR &First, sinhvien x)
{
  NODEPTR p, q;  // q la nut truoc, p la nut sau
  q = NULL;
  for(p = First; p != NULL && p->sv.mssv< x.mssv; p = p->next)
    q = p;
  if(q == NULL)  // them nut vao dau danh sach lien ket
    Insert_first(First, x);
  else          // them nut vao sau nut q
```

```
    Insert_after(q, x);
}
// Phep toan clearlist: xoa tat ca cac nut trong danh sach lien ket
void clearlist(NODEPTR &First)
{
  NODEPTR p, q;  // q la nut truoc, p la nut sau
  p = First;
  while(First != NULL)
  {
    p = First;
    First = First->next;
    free(p);
  }
}
// Phep toan traverse: duyet danh sach lien ket
void traverse(NODEPTR First)
{
  NODEPTR p;
  int stt = 0;
  p = First;
  if(p == NULL)
    printf("\n   (Khong co sinh vien trong danh sach)");
  while(p != NULL)
  {
    printf("\n %5d %8d %-30s %-10s", ++stt, p->sv.mssv, p->sv.ho,p->sv.ten);
    p = p->next;
  }
}
```

```
/* Tac vu search_info: tim kiem theo phuong phap tim kiem tuyen tinh, neu
khong tim thay ham nay tra ve NULL, neu tim thay ham nay tra ve con tro
chi  nut tim thay */
NODEPTR search_info(NODEPTR First, int x)
{
    NODEPTR p;
    p = First;
    while(p != NULL && p->sv.mssv != x  )
        p = p->next;
    return(p);
}
// Tac vu selectionsort: sap xep danh sach lien ket theo MSSV tang dan
void selectionsort(NODEPTR &First)
{
  NODEPTR p, q, pmin;
    sinhvien min;
    for(p = First; p->next != NULL; p = p->next)
    {
        min = p->sv;
        pmin = p;
        for(q = p->next; q != NULL; q = q->next)
            if(min.mssv > q->sv.mssv)
            {
                min = q->sv;
                pmin = q;
            }
        // hoan doi truong info cua hai nut p va pmin
        pmin->sv  = p->sv;
        p->sv   = min;
    }
```

```
}
char menu ()
{  char chucnang;
   do
   {  clrscr();
      printf("\n\n\t\tCHUONG TRINH QUAN LY DANH SACH SINH
VIEN");
      printf("\n\nCac chuc nang cua chuong trinh:\n");
      printf("   1: Tao danh sach sinh vien\n");
      printf("   2: Them sinh vien vao danh sach\n");
      printf("   3: Xem danh sach sinh vien\n");
      printf("   4: Hieu chinh sinh vien\n");
      printf("   5: Xoa sinh vien trong danh sach\n");
      printf("   6: Tim kiem sinh vien theo MSSV\n");
      printf("   7: Sap xep danh sach theo MSSV\n");
      printf("   8: Them sinh vien vao danh sach da co thu tu\n");
      printf("   9: Xoa toan bo danh sach\n");
      printf("   0: Ket thuc chuong trinh\n");
      printf("Chuc nang ban chon: ");
      chucnang = getche();
      } while(chucnang < '0' || chucnang >'9') ;
      return chucnang;
}
void Create_list(NODEPTR &First)
{ NODEPTR Last,p ;
  sinhvien sv;

  char maso [5],c;
      clearlist(First);
      printf("Ma so sinh vien: ");
```

```
      gets(maso);
      sv.mssv = atoi(maso);
  while (sv.mssv !=0)
  {
              printf("Ho sinh vien: ");
              gets(sv.ho);
              printf("Ten sinh vien: ");
              gets(sv.ten);
              p=New_node();
              p->sv=sv;
              if (First==NULL)
                First=p;
              else
                Last->next = p;
              Last=p;
              p->next=NULL;
              printf("Ma so sinh vien moi: ");
              gets(maso);
              sv.mssv = atoi(maso);
  }
}
// chuong trinh chinh
void main()
{
   int vitri;
   char chucnang, c, maso [5], c_vitri[5];
   // khoi dong danh sach lien ket
   initialize(First);
   do
   {
```

```
chucnang = menu();
flushall();
switch(chucnang)
{
case '1':
{
   Create_list(First);
   break;
}
case '2':
{
        printf("\nVi tri them (1, 2, ...): ");
        gets(c_vitri);
        vitri = atoi(c_vitri);
        p = nodepointer(First, vitri-1);//p chi nut truoc nut can them
        if (vitri <=0 || p==NULL)
        {
                printf("Vi tri khong hop le");
                getche();
        }
        else
        {
                printf("Ma so sinh vien: ");
                gets(maso);
                sv.mssv = atoi(maso);
                printf("Ho sinh vien: ");
                gets(sv.ho);
                printf("Ten sinh vien: ");
                gets(sv.ten);
                if (vitri == 1)
```

```
                Insert_first(First, sv);
            else
                Insert_after(p, sv);
        }
        break;
}
case '3':
{
        printf("\nDanh sach sinh vien: ");
        printf("\n     STT   MSSV        HO TEN");
        traverse(First);
        getche();
        break;
}
case '4':
{
        printf("\nVi tri hieu chinh (1, 2, ...): ");
        gets(c_vitri);
        vitri = atoi(c_vitri);
        p = nodepointer(First, vitri); // p chi nut can hieu chinh
        if(p == NULL)
        {
           printf("Vi tri khong phu hop");
          getche();
        }
        else
        {
          printf("\nSTT:%d  MSSV:%d  HO:%s          TEN:%s",
                vitri,p->sv.mssv, p->sv.ho, p->sv.ten);
          printf("\nMa so sv moi: ");
```

```
            gets(maso);
            sv.mssv = atoi(maso);
            printf("Ho sv moi: ");
            gets(sv.ho);
            printf("Ten sv moi: ");
            gets(sv.ten);
            p->sv = sv;
        }
break;
}
case '5':
{
        printf("\nVi tri xoa ( 1, 2, ...): ");
        gets(c_vitri);
        vitri = atoi(c_vitri);
        p = nodepointer(First, vitri-1);//p chi nut truoc nut can xoa
        if (vitri <=0 || p==NULL)
        {
                printf("Vi tri khong hop le");
                getche();
        }
        else
            if(vitri == 1)
               Delete_first(First);
            else
               Delete_after(p);
        break;
}
case '6':
{
```

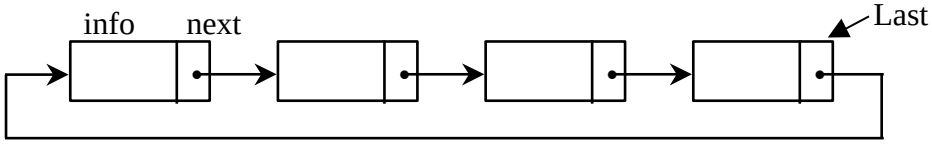
```
            printf("\nMa so sinh vien can tim: ");
            gets(maso);
            sv.mssv = atoi(maso);
            p = search_info(First, sv.mssv);
            if(p == NULL)
           printf("Khong co sinh vien co MSSV %d trong danh sach",
sv.mssv);
        else
        printf("Tim thay o vi tri %d trong danh sach", position(First, p));
        getche();
        break;
    }
    case '7':
    {
        printf("\n   Ban co chac khong? (c/k): ");
        c = toupper(getche());
        if( c == 'C')
            selectionsort(First);
        break;
    }
    case '8':
    {
       printf("\n   Ban nho sap xep danh sach truoc. Nhan phim bat ky ...");
       getche();
       printf("\nMa so sinh vien: ");
       gets(maso);
       sv.mssv = atoi(maso);
       printf("Ho sinh vien: ");
       gets(sv.ho);
       printf("Ten sinh vien: ");
```

```
            gets(sv.ten);
            Insert_Order(First, sv);
            break;
          }
          case '9':
          {
            printf("\n   Ban co chac khong (c/k): ");
            c = getche();
            if(c == 'c' || c == 'C')
                clearlist(First);
            break;
          }
          }
        } while(chucnang != '0');

        // xoa tat ca cac nut tren danh sach lien ket
          clearlist(First);
      }
```

IV. DANH SÁCH LIÊN KẾT VÒNG:

IV.1. Khái niệm :

Danh sách liên kết vòng là danh sách liên kết mà trường next của phần tử cuối sẽ chỉ tới phần tử đầu của danh sách.



Hình 4.8 Danh sách liên kết vòng

Qui ước: Để đơn giản giải thuật, ta qui ước dùng con trỏ Last để quản lý danh sách liên kết vòng, con trỏ này sẽ chỉ tới phần tử cuối trong danh sách liên kết vòng.

Như vậy:

+ Nếu danh sách liên kết vòng rỗng ⇔ Last = NULL

+ Nếu danh sách liên kết vòng chỉ có một phần tử ⇔ (Last ==Last->next)

- Khai báo: Ta khai báo biến Last quản lý danh sách liên kết vòng với thành phần nội dung là số nguyên như sau:

```
struct node
{  int info ;
   struct node *next ;
};
typedef struct node *NODEPTR;
NODEPTR Last;
```

IV.2. <u>Các phép toán trên danh sách liên kết vòng</u>:

IV.2.1.*<u>Tạo danh sách</u>*:

a. <u>Khởi tạo danh sách (Initialize)</u>: dùng để khởi động một danh sách liên kết, cho chương trình hiểu là hiện tại danh sách liên kết chưa có phần tử.

```
void Initialize(NODEPTR &Last)
{
  Last = NULL;
}
```

b. <u>Cấp phát vùng nhớ (New_Node)</u>: cấp phát một nút cho danh sách liên kết vòng. Hàm New_Node này trả về địa chỉ của nút vừa cấp phát.

```
NODEPTR New_node(void)
{
  NODEPTR p;
  p = (NODEPTR)malloc(sizeof(struct node));
  return (p);
}
```

c. <u>Thêm vào đầu danh sách (Ins_first)</u>: thêm một nút có nội dung x vào đầu danh sách liên kết vòng.

```
void Ins_first(NODEPTR &Last, int x)
{
  NODEPTR p;
```

```
  p = New_node();
  p->info = x;
  if (Empty(Last))
        Last=p;
  else
        p->next = Last->next;
  Last->next = p;
}
```

d. <u>Thêm vào cuối danh sách (Ins_last)</u>: thêm một nút có nội dung x vào cuối danh sách liên kết vòng.

```
void Ins_last(NODEPTR &Last, int x)
{
  NODEPTR p;
  p = New_node();
  p->info = x;
  if (Empty(Last))
        p->next=p;
  else
  {
        p->next = Last->next;
        Last->next = p;
  }
  Last = p;
}
```

e. <u>Thêm nút mới vào sau nút có địa chỉ p (Ins_after)</u>: thêm một nút có nội dung x vào sau nút có địa chỉ p trong danh sách liên kết vòng.

```
void Ins_after(NODEPTR Last, NODEPTR p, int x)
{
  NODEPTR q;
  if(p == NULL)
```

```
        printf("Nut hien tai khong co, nen khong the them");
  else
  {
    if (p==Last)
        Ins_last(Last,x);
    else
    {  q = New_node();
        q->info = x;
        q->next = p->next;
        p->next = q;
    }
  }
}
```

IV.2.2. *<u>Duyệt danh sách</u>*:

Thông thường ta hay duyệt danh sách liên kết để thực hiện một công việc gì đó, như liệt kê dữ liệu trong danh sách hay đếm số nút trong danh sách...

```
void Traverse(NODEPTR Last)
{
  NODEPTR p;
  p = Last->next;  // p chi toi phan tu dau trong dslk vong
  if(Last == NULL)
    printf("\n   Danh sach rong ");
  else
  {  printf("\n");
    while(p != Last)
    {
printf("%8d", p->info);
p = p->next;
    }
    printf("%8d",  p->info);
```

```
  }
}
```

IV.2.3. *Phép loại bỏ*:

a. Giải phóng vùng nhớ (free): Hàm này dùng để hủy nút đã cấp phát, và trả vùng nhớ về lại cho memory heap.

free(p) ; với p là biến con trỏ

b. Kiểm tra danh sách liên kết rỗng hay không (Empty): hàm Empty trả về TRUE nếu danh sách liên kết vòng rỗng, và ngược lại.

```
int Empty(NODEPTR Last)
{
  return(Last == NULL ? TRUE : FALSE);
}
```

c. Xóa phần tử đầu của danh sách (Del_first): muốn xóa 1 phần tử khỏi danh sách liên kết thì ta phải kiểm tra xem danh sách có rỗng hay không. Nếu danh sách có phần tử thì mới xóa được.

```
void Del_first(NODEPTR &Last)
{
  NODEPTR p;
  if(Empty(Last))
    printf("Khong co nut trong danh sach lien ket vong, nen khong the xoa");
  else
  {
    p = Last->next;    // nut can xoa la nut dau
    if (p==Last)     // danh sach chi co 1 nut
Last=NULL;
    else
Last->next = p->next;
    free(p);
  }
}
```

d. <u>Xóa phần tử cuối của danh sách (Del_last)</u>: muốn xóa 1 phần tử khỏi danh sách liên kết thì ta phải kiểm tra xem danh sách có rỗng hay không. Nếu danh sách có phần tử thì mới xóa được.

```
void Del_last(NODEPTR &Last)
{
  NODEPTR p;
  if(Empty(Last))
    printf("Khong co nut trong danh sach lien ket vong, nen khong the xoa");
  else
  {
    p = Last;    // nut can xoa la nut cuoi
    if (Last->next==Last)          // danh sach chi co 1 nut
       Last=NULL;
    else
    {
        for (NODEPTR q=Last->next;q->next !=Last; q=q->next);
       // q dung ngay truoc Last
        q->next = Last->next;
        Last=q;
    }
    free(p);
  }
}
```

e. <u>Xóa phần tử đứng sau nút có địa chỉ p (Del_after)</u>: xóa nút sau nút p. Phép toán này không xóa được khi danh sách đã rỗng hoặc danh sách chỉ có 1 nút

```
void Del_after(NODEPTR &Last, NODEPTR p)
{
  NODEPTR q;
  if(Empty(Last))
    printf("Khong co nut trong danh sach lien ket vong, nen khong the xoa");
```

```
  else
  { // neu p la NULL hoac danh sach chi co 1 nut
   if((p == NULL) || (Last->next == Last))
       printf("khong the xoa trong danh sach lien ket vong duoc");
   else
   {
    q=p->next;
    if (p->next == Last)
    {
       p->next=Last->next;
       Last=p;
    }
    else
       p->next=q->next;
    free(q);
   }
  }
}
```

f. <u>Xóa toàn bộ danh sách (Clearlist)</u>: ta có thể sử dụng lệnh Last=NULL để xóa toàn bộ danh sách, nhưng trong bộ nhớ, các vùng nhớ đã cấp phát cho các nút không giải phóng về lại cho memory heap, nên sẽ lãng phí vùng nhớ. Do đó, ta sử dụng giải thuật sau:

```
void clearlist(NODEPTR &Last)
{
  while(Last != NULL)
    Del_first(Last);
}
```

IV.2.4. ***<u>Tìm kiếm (Srch_info)</u>*** **:**

Tìm nút đầu tiên trong danh sách liên kết vòng có info bằng với x.

Hàm Srch_info nếu tìm thấy x trong danh sách thì trả về địa chỉ của nút đó trong danh sách, nếu không có thì trả về trị NULL.

```
NODEPTR Srch_info(NODEPTR Last, int x)
{
  NODEPTR p;
  if (Empty(Last))
     return (NULL);
  else
  {
     p = Last->next;  // p chi toi phan tu dau cua dslk vong
     if (p->info==x)
       return (p);
     else
     {  p=p->next;
while(p != Last->next && p->info != x  )
                  p = p->next;
return (p->info==x ? p : NULL);
     }
  }
}
```

IV.2.5. *<u>Sắp xếp (Selection_Sort)</u>*:

Sắp xếp danh sách liên kết vòng theo thứ tự info tăng dần theo phương pháp Selection sort.

- <u>Nội dung</u>: Ta so sánh tất cả các phần tử của danh sách để chọn ra một phần tử nhỏ nhất đưa về đầu danh sách; sau đó, tiếp tục chọn phần tử nhỏ nhất trong các phần tử còn lại để đưa về phần tử thứ hai trong danh sách. Quá trình này lặp lại cho đến khi chọn ra được phần tử nhỏ thứ (n-1).

- <u>Giải thuật:</u>

```
void selectionsort(NODEPTR &Last)
{
   NODEPTR p, q, pmin;
```

```
        int min;
        for(p = Last->next; p->next != Last->next; p = p->next)
        {
                min = p->info;
                pmin = p;
                for(q = p->next; q != Last->next; q = q->next)
                        if(min > q->info)
                        {
                                min = q->info;
                                pmin = q;
                        }
                // hoan doi truong info cua hai nut p va pmin
                pmin->info  = p->info;
                p->info     = min;
        }
}
```

<u>Ví dụ</u>: Viết chương trình thực hiện các công việc sau trên một danh sách các số nguyên với cấu trúc dữ liệu là danh sách liên kết vòng :

1. Tạo danh sách số
2. Thêm phần tử vào đầu danh sách
3. Thêm phần tử vào cuối danh sách
4. Thêm phần tử vào sau phần tử có giá trị x
5. Xóa phần tử đầu trong danh sách
6. Xóa phần tử cuối trong danh sách
7. Liệt kê danh sách
8. Sắp xếp danh sách theo thứ tự tăng
9. Xóa toàn bộ danh sách

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
```

```
#include <alloc.h>
#include <ctype.h>
#define TRUE 1
#define FALSE 0
struct node
{
  int info;
  struct node *next;
};
typedef struct node *NODEPTR;
 NODEPTR Last;
// Phep toan New_node: cap phat mot nut cho danh sach lien ket
NODEPTR New_node(void)
{
  NODEPTR p;
  p = (NODEPTR)malloc(sizeof(struct node));
  return(p);
}
// Phep toan initialize: khoi dong danh sach lien ket
void Initialize(NODEPTR &Last)
{
  Last = NULL;
}
// Tac vu Empty: kiem tra danh sach lien ket co bi rong khong
int Empty(NODEPTR Last)
{
  return(Last == NULL ? TRUE : FALSE);
}
// Phep toan Ins_first: them nut moi vao dau danh sach lien ket vong
void Ins_first(NODEPTR &Last, int x)
```

```
{
  NODEPTR p;
  p = New_node();
  p->info = x;
  if (Empty(Last))
        Last=p;
  else
        p->next = Last->next;
  Last->next = p;
}
// Phep toan Ins_last: them nut moi vao cuoi danh sach lien ket vong
void Ins_last(NODEPTR &Last, int x)
{
  NODEPTR p;
  p = New_node();
  p->info = x;
  if (Empty(Last))
        p->next=p;
  else
  {
        p->next = Last->next;
        Last->next = p;
  }
  Last = p;
}
// Phep toan Ins_after: them nut moi sau nut co dia chi p
void Ins_after(NODEPTR Last, NODEPTR p, int x)
{
  NODEPTR q;
  if(p == NULL)
```

```
        printf("Nut hien tai khong co, nen khong the them");
  else
  {
    if (p==Last)
        Ins_last(Last,x);
    else
    {  q = New_node();
        q->info = x;
        q->next = p->next;
        p->next = q;
    }
  }
}
// Phep toan Del_first: xoa nut o dau danh sach lien ket
void Del_first(NODEPTR &Last)
{
  NODEPTR p;
  if(Empty(Last))
    printf("Khong co nut trong danh sach lien ket vong, nen khong the xoa");
  else
  {
    p = Last->next;    // nut can xoa la nut dau
    if (p==Last)     // danh sach chi co 1 nut
        Last=NULL;
    else
        Last->next = p->next;
    free(p);
  }
}
// Phep toan Del_last: xoa nut o cuoi danh sach lien ket
```

```
void Del_last(NODEPTR &Last)
{
  NODEPTR p;
  if(Empty(Last))
    printf("Khong co nut trong danh sach lien ket vong, nen khong the xoa");
  else
  {
    p = Last;    // nut can xoa la nut cuoi
    if (Last->next==Last)          // danh sach chi co 1 nut
        Last=NULL;
    else
    {
         for (NODEPTR q=Last->next;q->next !=Last; q=q->next);
        // q dung ngay truoc Last
         q->next = Last->next;
         Last=q;
    }
    free(p);
  }
}
// Tac vu Del_after: xoa nut sau nut p.  Phep toan nay khong xoa duoc
// khi da rong hoac ds chi co 1 nut
void Del_after(NODEPTR &Last, NODEPTR p)
{
  NODEPTR q;
  if(Empty(Last))
    printf("Khong co nut trong danh sach lien ket vong, nen khong the xoa");
  else
  { // neu p la NULL hoac danh sach chi co 1 nut
   if((p == NULL) || (Last->next == Last))
```

```
        printf("khong the xoa trong danh sach lien ket vong duoc");
    else
    {
      q=p->next;
      if (p->next == Last)
      {
         p->next=Last->next;
         Last=p;
      }
      else
         p->next=q->next;
      free(q);
    }
  }
}
// Phep toan clearlist: xoa tat ca cac nut trong danh sach lien ket vong
void clearlist(NODEPTR &Last)
{
  while(Last != NULL)
     Del_first(Last);
}
// Phep toan traverse: duyet danh sach lien ket vong
void traverse (NODEPTR Last)
{
  NODEPTR p;
  p = Last->next;  // p chi toi phan tu dau trong dslk vong
  if(Last == NULL)
     printf("\n   Danh sach rong ");
  else
  {  printf("\n");
```

```
    while(p != Last)
    {
        printf("%8d", p->info);
        p = p->next;
    }
    printf("%8d",  p->info);
  }
}
```

/* Phép toán Srch_info: tim kiem theo phuong phap tim kiem tuyen tinh, neu khong tim thay ham nay tra ve NULL, neu tim thay ham nay tra ve con tro chi nut tim thay */

```
NODEPTR Srch_info(NODEPTR Last, int x)
{
 NODEPTR p;
 if (Empty(Last))
   return (NULL);
 else
 {
   p = Last->next;  // p chi toi phan tu dau cua dslk vong
   if (p->info==x)
     return (p);
   else
   {  p=p->next;
       while(p != Last->next && p->info != x  )
              p = p->next;
       return (p->info==x ? p : NULL);
   }
 }
}
```

// Tac vu selectionsort: sap xep danh sach lien ket vong theo info tang dan

```
void selectionsort(NODEPTR &Last)
{
  NODEPTR p, q, pmin;
        int min;
        for(p = Last->next; p->next != Last->next; p = p->next)
        {
                min = p->info;
                pmin = p;
                for(q = p->next; q != Last->next; q = q->next)
                        if(min > q->info)
                        {
                                min = q->info;
                                pmin = q;
                        }
                // hoan doi truong info cua hai nut p va pmin
                pmin->info  = p->info;
                p->info     = min;
        }
}
void Create_list(NODEPTR &Last)
{
 int nd;
        clearlist(Last);
        printf("Nhap so (ket thuc bang 0: ");
        scanf("%d", &nd);
 while (nd !=0)
 {
        Ins_last(Last, nd);
        printf("Nhap so ke : ");
        scanf("%d", &nd);
```

```
  }
}
char menu ()
{  char chucnang;
   do
   {  clrscr();
      printf("\n\n\t\tCHUONG TRINH QUAN LY DANH SACH LIEN KET VONG");
      printf("\n\nCac chuc nang cua chuong trinh:\n");
      printf("   1: Tao danh sach \n");
      printf("   2: Them phan tu vao dau danh sach\n");
      printf("   3: Them phan tu vao cuoi danh sach\n");
      printf("   4: Them phan tu vao sau phan tu co gia tri x\n");
      printf("   5: Xoa phan tu dau trong danh sach \n");
      printf("   6: Xoa phan tu cuoi trong danh sach \n");
      printf("   7: Liet ke danh sach \n");
      printf("   8: Sap xep danh sach theo thu tu tang\n");
      printf("   9: Xoa toan bo danh sach\n");
      printf("   0: Ket thuc chuong trinh\n");
      printf("Chuc nang ban chon: ");
      chucnang = getche();
      } while(chucnang < '0' || chucnang >'9') ;
      printf("\n");
      return chucnang;
}
// chuong trinh chinh
void main()
{
   int  x, info;
   char chucnang, c;
   NODEPTR p;
```

```
// khoi dong danh sach lien ket
Initialize(Last);
do
{
  chucnang = menu();
  switch(chucnang)
  {
          case '1':
          {
             Create_list(Last);
             break;
          }
          case '2':
          {
             printf("\nNoi dung muon them: ");
             scanf("%d",&x);
             Ins_first(Last,x);
             break;
          }
          case '3':
          {
             printf("\nNoi dung muon them: ");
             scanf("%d",&x);
             Ins_last(Last,x);
             break;
          }
          case '4':
          {
             printf("\nNoi dung phan tu muon them: ");
             scanf("%d",&x);
```

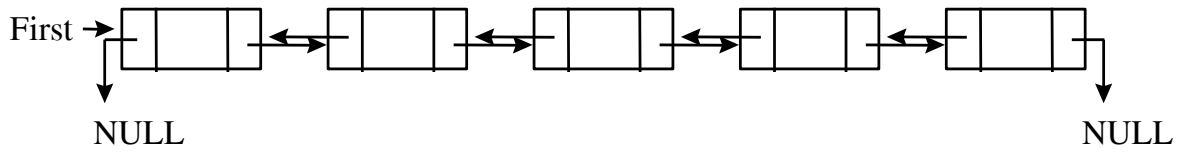
```
    printf("\nBan muon them no vao sau phan tu co info = ");
    scanf("%d",&info);
    p=Srch_info(Last,info);
    if (p==NULL)
    {
      printf("Khong co phan tu voi x=%d", info);
      getch();
    }
    else
      if (p==Last)
        Ins_last(Last,x);
      else
        Ins_after(Last,p,x);
    break;
}
case '5':
     Del_first(Last);
     break;
case '6':
     Del_last(Last);
     break;
case '7':
{
     Traverse(Last);
     getche();
     break;
 }
 case '8':
 {
     printf("\n   Ban co chac khong? (c/k): ");
```

```
            c = toupper(getche());
            if( c == 'C')
              selectionsort(Last);
            break;
        }
        case '9':
        {
            printf("\n   Ban co chac khong (c/k): ");
            c = getche();
            if(c == 'c' || c == 'C')
              clearlist(Last);
            break;
        }
    }
  } while(chucnang != '0');
  // xoa tat ca cac nut tren danh sach lien ket
      clearlist(Last);
}
```

V. DANH SÁCH LIÊN KẾT KÉP (Doubly Linked List) :

V.1. Khái niệm :

Danh sách liên kết kép là một danh sách liên kết mà mỗi phần tử của nó có 2 vùng liên kết, liên kết thuận dùng để chỉ đến phần tử đứng ngay sau nó (right), liên kết nghịch dùng để chỉ đến phần tử đứng ngay trước nó (left).



Hình 4.9 Danh sách liên kết kép

Lưu ý:

- Nút cuối của danh sách liên kết kép có trường right là NULL, và nút đầu của

danh sách liên kết kép có trường left là NULL.

- Chúng ta có thể duyệt danh sách liên kết kép theo hai chiều duyệt xuôi và duyệt ngược.

* Khai báo: Ta khai báo biến First quản lý danh sách liên kết kép với thành phần nội dung là số nguyên như sau:

```
struct node
{  int info ;
   struct node *left, *right ;
};
typedef struct node *NODEPTR;
NODEPTR First;
```

* Khởi tạo :

```
First = NULL ;
```

- Biến First : con trỏ chỉ đến phần tử đầu danh sách liên kết kép

V.2. Các phép toán trên danh sách liên kết kép:

V.2.1.*Tạo danh sách*:

a. Khởi tạo danh sách (Initialize): dùng để khởi động một danh sách liên kết, cho chương trình hiểu là hiện tại danh sách liên kết chưa có phần tử.

```
void Initialize(NODEPTR &First)
{
  First = NULL;
}
```

b. Cấp phát vùng nhớ (New_Node): cấp phát một nút cho danh sách liên kết kép. Hàm New_Node này trả về địa chỉ của nút vừa cấp phát.

```
NODEPTR New_node(void)
{
  NODEPTR p;
  p = (NODEPTR)malloc(sizeof(struct node));
  return (p);
}
```

c. <u>Thêm vào đầu danh sách (Insert_first)</u>: thêm một nút có nội dung x vào đầu danh sách liên kết kép.

```
void Insert_first(NODEPTR &First, int x)
{
  NODEPTR p;
  p = New_node();
  p->info = x;
  if(First == NULL)  // truong hop danh sach rong
    p->right = NULL;
  else
  {
    // tao lien ket giua p va First
    p->right = First;
    First->left = p;
  }
   First = p;
   p->left = NULL;
}
```

d. <u>Thêm nút mới vào sau nút có địa chỉ p (Insert_right)</u>: thêm một nút có nội dung x vào sau nút có địa chỉ p trong danh sách liên kết kép. Phép toán này cũng được dùng để thêm một nút vào cuối danh sách.

```
void Insert_right(NODEPTR p, int x)
{
  NODEPTR q, r;  // q la nut can them vao, p la nut truoc, r la nut sau
  if(p == NULL)
    printf("Nut p khong hien huu, khong them nut duoc\n");
  else
  {
    q = New_node();
    q->info = x;
```

```
      r = p->right;
      // tao hai lien ket giua r va q
      r->left = q;
      q->right = r;
      // tao hai lien ket giua p va q
      q->left = p;
      p->right = q;
    }
  }
```

e. <u>Thêm nút mới vào trước nút có địa chỉ p (Insert_left)</u>: thêm một nút có nội dung x vào trước nút có địa chỉ p trong danh sách liên kết kép.

```
void Insert_left(NODEPTR &First, NODEPTR p, int x)
  {
    NODEPTR q, r;  // q la nut can them vao, p la nut sau, r la nut truoc
    if(p == NULL)
      printf("Nut p khong hien huu, khong them nut duoc\n");
    else
    {
      if(p == First)  // them nut vao dau danh sach
        Insert_first(First, x);
      else
      {
        q = New_node();
        q->info = x;
        r = p->left;
        // tao hai lien ket giua r va q
        r->right = q;
        q->left = r;
        // tao hai lien ket giua p va q
        q->right = p;
```

```
            p->left = q;
          }
        }
      }
```

V.2.2. *Duyệt danh sách*:

Thông thường ta hay duyệt danh sách liên kết để thực hiện một công việc gì đó, như liệt kê dữ liệu trong danh sách hay đếm số nút trong danh sách...

a. Duyệt xuôi: Duyệt danh sách liên kết kép từ nút đầu cho tới nút cuối danh sách.

```
void Right_traverse(NODEPTR First)
{
  NODEPTR p;
   if(empty(First))
     printf("\n  (khong co doan nao)");
  else
  {
     p = First;  // p chi nut dau
     while(p != NULL)
     {
          printf("\n%8d, p->info);
          p = p->right;
     }
   }
 }
```

b. Duyệt ngược: Duyệt danh sách liên kết kép từ nút cuối cho tới nút đầu danh sách.

```
void Left_traverse(NODEPTR First)
{
  NODEPTR p;
  if(empty(First))
    printf("\n  (khong co doan nao)");
```

```
  else
  {
    for (p=First; p->right!=NULL; p=p->right); // p chi toi nut cuoi
    while(p != NULL)
    {
      printf("\n%8d, p->info
      p = p->left;
    }
  }
}
```

V.2.3. *<u>Phép loại bỏ</u>*:

a. <u>Giải phóng vùng nhớ(free)</u>: Hàm này dùng để hủy nút đã cấp phát, và trả vùng nhớ về lại cho memory heap.

free(p) ; với p là biến con trỏ

b. <u>Kiểm tra danh sách liên kết rỗng hay không (Empty)</u>: hàm Empty trả về TRUE nếu danh sách liên kết vòng rỗng, và ngược lại.

```
int Empty(NODEPTR First)
{
  return(First == NULL ? TRUE : FALSE);
}
```

c. <u>Xóa phần tử đầu của danh sách (Delete_first)</u>: muốn xóa 1 phần tử khỏi danh sách liên kết thì ta phải kiểm tra xem danh sách có rỗng hay không; nếu danh sách có phần tử thì mới xóa được.

```
void Delete_first(NODEPTR &First)
{
  NODEPTR p;
  if(empty(First))  // truong hop danh sach rong
    printf("Danh sach rong, khong xoa nut duoc");
  else
  {
```

```
      p = First;  // p la nut can xoa
      if(First->right == NULL)  // truong hop danh sach co mot nut
        First = NULL;
      else
      {
        First = p->right;
        First->left = NULL;
      }
      free(p);
    }
  }
```

d. <u>Xóa phần tử có địa chỉ p (Delete_node)</u>:

```
  void Delete_node(NODEPTR &First, NODEPTR p)
  {
    NODEPTR q, r;
    if(p == NULL)
      printf("Nut p khong hien huu, khong xoa nut duoc\n");
    else
    {
      if(First == NULL)  // truong hop danh sach rong
        printf("Danh sach rong, khong xoa nut duoc");
      else
      {
        if(p == First)  // truong hop xoa nut dau
          Delete_first(First);
        else
        {
          q = p->left;   // q la nut truoc
          r = p->right;  // r la nut sau
          // tao hai lien ket giua q va r
```

```
            r->left = q;
            q->right = r;
            free(p);
          }
        }
      }
    }
```

e. <u>Xóa toàn bộ danh sách (Clearlist)</u>:

```
    void clearlist(NODEPTR &First)
    {
      while(First != NULL)
        Delete_first(Last);
    }
```

V.2.4. *<u>Tìm kiếm</u>* (Search_info):

Tìm nút đầu tiên trong danh sách liên kết kép có info bằng với x.

Hàm Search_info nếu tìm thấy x trong danh sách thì trả về địa chỉ của nút đó trong danh sách, nếu không có thì trả về trị NULL.

```
NODEPTR Search_info(NODEPTR First, int x)
{
  NODEPTR p;
  p = First;
  while(p->info != x && p != NULL)
    p = p->right;
  return(p);
}
```

<u>Ví dụ</u>:

Viết chương trình quản lý và điều hành tuyến xe lửa TP HCM - HA NOI bằng danh sách liên kết kép; mỗi nút của danh sách là một đoạn đường có ga trước, ga sau, chiều dài và thời gian xe lửa chạy trên đoạn đường đó.

Chương trình có các chức năng sau:

1. Thêm một đoạn đường
2. Xóa một đoạn đường
3. Xem toàn tuyến đường theo liên kết xuôi
4. Xem toàn tuyến đường theo liên kết ngược
5. Xem thông tin của đoạn đường thứ i
6. Hiệu chỉnh thông tin của đoạn đường thứ i
7. Báo lộ trình: nhập nơi đi và nơi đến, chương trình sẽ cho biết các ga trung gian phải đi qua, tổng chiều dài và tổng thời gian của lộ trình.

- Chương trình:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <alloc.h>
#include <dos.h>
#include <ctype.h>
#define TRUE 1
#define FALSE 0
// Khai bao cau truc cua mot doan duong tren tuyen duong
typedef struct doan
{
  char gatruoc[12];
  char gasau[12];
  int chieudai;   // km
  int thoigian;   // thoi gian xe lua chay tren doan, tinh theo phut
};
// Khai bao cau truc cua mot nut
struct node
{
  doan info;
```

```
  struct node *left, *right;
};
typedef struct node *NODEPTR;
// Tac vu New_node: cap phat mot nut cho danh sach lien ket kep
NODEPTR New_node(void)
{
  NODEPTR p;
  p = (NODEPTR)malloc(sizeof(struct node));
  return(p);
}
// Tac vu initialize: khoi dong danh sach lien ket kep
void Initialize(NODEPTR &First)
{
  First = NULL;
}
// Tac vu empty: kiem tra danh sach lien ket kep co bi rong khong
int empty(NODEPTR First)
{
  return((First == NULL) ? TRUE : FALSE);
}
// Tac vu Insert_first: them nut vao dau danh sach lien ket
void Insert_first(NODEPTR &First, doan x)
{
  NODEPTR p;
  p = New_node();
  p->info = x;
  if(First == NULL)  // truong hop danh sach rong
    p->right = NULL;
  else
  {
```

```
      // tao lien ket giua p va First
      p->right = First;
      First->left = p;
}
     First = p;
     p->left = NULL;
}
/ Tac vu Insert_right: them nut moi sau nut p
void Insert_right(NODEPTR p, doan x)
{
   NODEPTR q, r;  // q la nut can them vao, p la nut truoc, r la nut sau
   if(p == NULL)
      printf("Nut p khong hien huu, khong them nut duoc\n");
   else
   {
      q = New_node();
      q->info = x;
      r = p->right;
      // tao hai lien ket giua r va q
      r->left = q;
      q->right = r;
      // tao hai lien ket giua p va q
      q->left = p;
      p->right = q;
   }
}
// Tac vu Insert_left: them nut moi truoc nut p
void Insert_left(NODEPTR &First, NODEPTR p, doan x)
{
   NODEPTR q, r;  // q la nut can them vao, p la nut sau, r la nut truoc
```

```
  if(p == NULL)
    printf("Nut p khong hien huu, khong them nut duoc\n");
  else
  {
    if(p == First)  // them nut vao dau danh sach
      Insert_first(First, x);
    else
    {
      q = New_node();
      q->info = x;
      r = p->left;
      // tao hai lien ket giua r va q
      r->right = q;
      q->left = r;
      // tao hai lien ket giua p va q
      q->right = p;
      p->left = q;
    }
  }
}
// Tac vu Delete_first: xoa nut o dau danh sach lien ket
void Delete_first(NODEPTR &First)
{
  NODEPTR p;
  if(empty(First))  // truong hop danh sach rong
    printf("Danh sach rong, khong xoa nut duoc");
  else
  {
    p = First;  // p la nut can xoa
    if(First->right == NULL)  // truong hop danh sach co mot nut
```

```
        First = NULL;
      else
      {
        First = p->right;
        First->left = NULL;
      }
      free(p);
    }
}
// Tac vu Delete_node: xoa nut co con tro la p
void Delete_node(NODEPTR &First, NODEPTR p)
{
  NODEPTR q, r;
  if(p == NULL)
    printf("Nut p khong hien huu, khong xoa nut duoc\n");
  else
  {
    if(First == NULL)  // truong hop danh sach rong
      printf("Danh sach rong, khong xoa nut duoc");
    else
    {
      if(p == First)  // truong hop xoa nut dau
         Delete_first(First);
      else
      {
        q = p->left;   // q la nut truoc
        r = p->right;  // r la nut sau
        // tao hai lien ket giua q va r
        r->left = q;
        q->right = r;
```

```
            free(p);
          }
        }
      }
}
// Tac vu Right_traverse: duyet danh sach tu trai sang phai (duyet xuoi)
void Right_traverse(NODEPTR First)
{
  NODEPTR p;
  int stt;
  if(empty(First))
    printf("\n  (khong co doan nao)");
  else
  {
    p = First;  // p chi nut dau
    stt = 1;
    while(p != NULL)
    {
              printf("\n%5d%12s%12s%7d%7d", stt++, p->info.gatruoc,
          p->info.gasau, p->info.chieudai, p->info.thoigian);
        p = p->right;
    }
  }
}
// Tac vu Left_traverse: duyet danh sach tu phai sang trai (duyet nguoc)
void Left_traverse(NODEPTR First)
{
  NODEPTR p;
  int stt;
  if(empty(First))
```

```
    printf("\n  (khong co doan nao)");
  else
  {
    for (p=First; p->right!=NULL; p=p->right); // p chi toi nut cuoi
    stt = 1;
    while(p != NULL)
    {
      printf("\n%5d%12s%12s%7d%7d", stt++, p->info.gasau,
       p->info.gatruoc, p->info.chieudai, p->info.thoigian);
      p = p->left;
    }
  }
}
// Tac vu Search_info1: tim ga truoc cua mot doan
NODEPTR Search_info1(NODEPTR First, char x[])
{
  NODEPTR p;
  p = First;
  while(strcmp(p->info.gatruoc, x) != 0 && p != NULL)
    p = p->right;
  return(p);
}
// Tac vu Search_info2: tim ga sau cua mot doan
NODEPTR Search_info2(NODEPTR First, char x[])
{
  NODEPTR p;
  p = First;
  while(strcmp(p->info.gasau, x) != 0 && p != NULL)
    p = p->right;
  return(p);
```

```
}
// Tac vu clearlist: xoa toan bo danh sach lien ket kep
void clearlist(NODEPTR &First)
{
  while(First != NULL)
    Delete_first(First);
}
int position(NODEPTR First, NODEPTR p)
{
      int vitri;
  NODEPTR q;
      q = First;
      vitri = 0;
      while(q != NULL && q != p)
      {
            q = q->right;
            vitri++;
      }
  if(q == NULL)
    return(-1);
      return(vitri);
}
void baolotrinh(NODEPTR &First, char noidi[], char noiden[], char c)
{
  NODEPTR p1, p2;
  int kc, tg;
  if(c == 'X')
  {
        p1 = Search_info1(First, noidi);
        if(p1 == NULL)
```

```
 {
    printf("Khong co noi di");
    return;
 }
if(strcmp(noidi, noiden) == 0)
{
   printf("Noi di trung noi den");
   return;
}
p2 = Search_info2(First, noiden);
if(p2 == NULL)
{
    printf("Khong co noi den");
    return;
}
if(position(First, p1) <= position(First, p2))
{
  kc = tg = 0;
  while(p1 != p2)
{
    kc = kc + p1->info.chieudai;
    tg = tg + p1->info.thoigian;
    printf("\n%s -> %s :  %d km  %d phut", p1->info.gatruoc,
           p1->info.gasau, p1->info.chieudai, p1->info.thoigian);
    p1 = p1->right;
}
kc = kc + p1->info.chieudai;
tg = tg + p1->info.thoigian;
printf("\n%s -> %s :  %d km  %d phut", p1->info.gatruoc,
         p1->info.gasau, p1->info.chieudai, p1->info.thoigian);
```

```
        printf("\nTong chieu dai lo trinh: %d km. Tong thoi gian van  chuyen
%d phut", kc, tg);
    }
    else
       printf("Khong di xuoi duoc");
    return;
  }

  if(c == 'N')
  {
        p1 = Search_info2(First, noidi);
        if(p1 == NULL)
        {
          printf("Khong co noi di");
          return;
        }
        if(strcmp(noidi, noiden) == 0)
        {
         printf("Noi di trung noi den");
         return;
        }
        p2 = Search_info1(First, noiden);
        if(p2 == NULL)
        {
              printf("Khong co noi den");
              return;
        }
       if(position(First, p1) >= position(First, p2))
       {
        kc = tg = 0;
```

```
            while(p1 != p2)
            {
               kc = kc + p1->info.chieudai;
               tg = tg + p1->info.thoigian;
               printf("\n%s -> %s :  %d km  %d phut", p1->info.gasau,
                     p1->info.gatruoc, p1->info.chieudai, p1->info.thoigian);
               p1 = p1->left;
             }
             kc = kc + p1->info.chieudai;
             tg = tg + p1->info.thoigian;
             printf("\n%s -> %s :  %d km  %d phut", p1->info.gasau,
                  p1->info.gatruoc, p1->info.chieudai, p1->info.thoigian);
             printf("\nTong chieu dai lo trinh: %d km. Tong thoi gian van
                       chuyen %d phut", kc, tg);
      }
      else
         printf("Khong di nguoc duoc");
      return;
   }
}
/* Tac vu nodepointer: xac dinh con tro chi nut thu i (i=0,1,2,...) trong
   danh sach lien ket kep */
NODEPTR nodepointer(NODEPTR First, int i)
{
   NODEPTR p;
   int vitri;
   p = First;      // p chi nut dau dslk vong
   vitri = 1;
   while(p != NULL && vitri < i)
   {
```

```
    p = p->right;
    vitri++;
  }
  return(p);
}
char menu ()
{  char chucnang;
  do
  {  clrscr();
    // menu chinh cua chuong trinh
    printf("\n\nCHUONG TRINH QUAN LY VA DIEU HANH
                TUYEN XE LUA TPHCM - HANOI\n");
    printf("   1: Them mot doan\n");
    printf("   2: Xoa mot doan\n");
    printf("   3: Xem lo trinh 1 (duyet xuoi)\n");
    printf("   4: Xem lo trinh 2 (duyet nguoc)\n");
    printf("   5: Xem thong tin cua doan thu i\n");
    printf("   6: Hieu chinh thong tin ve doan thu i\n");
    printf("   7: Bao lo trinh\n");
    printf("   0: Ket thuc chuong trinh\n");
    printf("Chuc nang ban chon: ");
    chucnang = getche();
    } while(chucnang < '0' || chucnang >'7') ;
    return chucnang;
}
// chuong trinh chinh
void main()
{
  NODEPTR First, p, p1;
  doan ga;
```

```
int vitri;
char c, chucnang;
char noidi[12], noiden[12];
char c_vitri[3], c_chieudai[10], c_thoigian[10];
clrscr();
// khoi dong danh sach lien ket kep
Initialize(First);
do
{
   chucnang=menu();
   switch(chucnang)
  {
     case '1':
         {
            printf("\nVi tri them (1, 2, ...): ");
            gets(c_vitri);
            vitri = atoi(c_vitri);
            p = nodepointer(First, vitri-1);//p chi nut truoc nut can them
            if (vitri <=0 || (p==NULL && First !=NULL))
            {
                    printf("Vi tri khong hop le");
                    getche();
            }
            else
            {
             printf("Ten ga truoc: ");
             gets(ga.gatruoc);
             printf("Ten ga sau: ");
             gets(ga.gasau);
             printf("Chieu dai (km): ");
```

```
    gets(c_chieudai);
    ga.chieudai = atoi(c_chieudai);
    printf("Thoi gian (phut): ");
    gets(c_thoigian);
    ga.thoigian = atoi(c_thoigian);
    if(vitri == 1 || First ==NULL)
      Insert_first(First, ga);
    else
      Insert_right(p, ga);
}
    break;
}
case '2':
{
    printf("\nVi tri muon xoa(1,2,...): ");
    gets(c_vitri);
    vitri = atoi(c_vitri);
    p = nodepointer(First, vitri);
    if(p == NULL)
      printf("Vi tri khong hop le");
    else
    {
      if(vitri == 1)
        Delete_first(First);
      else
        Delete_node(First, p);
      printf("Da xoa xong ");
    }
    delay(2000);
    break;
```

```
}
case '3':
{
    printf("\nXem lo trinh 1 (duyet xuoi): ");
    printf("\n  STT        TU        DEN     CD     TG");
    Right_traverse(First);
    getche();
    break;
}
case '4':
{
    printf("\nXem lo trinh 2 (duyet nguoc): ");
    printf("\n  STT        TU        DEN     CD     TG");
    Left_traverse(First);
    getche();
    break;
}
case '5':
{
    printf("\nVi tri doan muon xem thong tin(1,2,...): ");
    gets(c_vitri);
    vitri = atoi(c_vitri);
    p = nodepointer(First, vitri);
    if(p == NULL)
      printf("Vi tri khong hop le");
    else
       printf("\nDoan:%d  Tu:%s  Den:%s  Chieu dai:%d km
                  Thoi gian:%d phut", vitri, p->info.gatruoc,
                  p->info.gasau, p->info.chieudai, p->info.thoigian);
    getche();
```

```
    break;
}
case '6':
{
    printf("\nVi tri doan muon hieu chinh(1,2,...): ");
    gets(c_vitri);
    vitri = atoi(c_vitri);
    p = nodepointer(First, vitri);
    if(p == NULL)
      printf("Vi tri khong hop le");
    else
    {
      printf("\nDoan:%d  Tu:%s  Den:%s  Chieu dai:%d km
            Thoi gian:%d phut\n", vitri, p->info.gatruoc,
            p->info.gasau, p->info.chieudai, p->info.thoigian);
      printf("Ten ga truoc: ");
      gets(ga.gatruoc);
      printf("Ten ga sau: ");
      gets(ga.gasau);
      printf("Chieu dai (km): ");
      gets(c_chieudai);
      ga.chieudai = atoi(c_chieudai);
      printf("Thoi gian (phut): ");
      gets(c_thoigian);
      ga.thoigian = atoi(c_thoigian);
      p->info = ga;
    }
    break;
}
case '7':
```

```
            {
               printf("\nBan di xuoi hay nguoc (x/n): ");
               c = toupper(getch());
               printf("\nCho biet noi di: ");
               gets(noidi);
               printf("\nCho biet noi den: ");
               gets(noiden);
               baolotrinh(First, noidi, noiden, c);
               getch();
               break;
            }
         }
      } while(chucnang != '0');
      // Xoa toan bo cac nut tren danh sach lien ket kep
      clearlist(First);
   }
```

2.3. Các kiểu dữ liệu trừu tượng

2.3.1 Stack:

2.3.1.1 *Khái niệm*:

Như ta đã biết, Stack là một danh sách mà việc thêm vào và loại bỏ một phần tử chỉ diễn ra cùng một đầu của danh sách, tức là theo cơ chế LIFO (Last In First Out). Trong chương 3, ta đã khảo sát Stack với cấu trúc dữ liệu là danh sách tuyến tính, việc thêm vào và loại bỏ diễn ra ở cuối danh sách. Với danh sách tuyến tính làm Stack thì Stack có điểm hạn chế về số lượng phần tử phải khai báo trước. Để khắc phục nhược điểm này, ta sẽ xây dựng Stack với cấu trúc dữ liệu là danh sách liên kết đơn. Việc thêm vào và loại bỏ sẽ diễn ra ở đầu danh sách.

- Khai báo: Ta khai báo biến sp (Stack Pointer) là con trỏ chỉ đến một danh sách là Stack, mỗi phần tử trong Stack là 1 số nguyên như sau:

```
struct node
{  int info ;
   struct node *next ;
```

```
};
typedef struct node *Stack;
Stack sp;
```

- Khởi tạo Stack : sp = NULL;

Lưu ý:

- Với Stack là danh sách liên kết, ta chỉ thực hiện các phép toán ở đầu danh sách liên kết.

- Không có trường hợp Stack đầy ; Stack rỗng khi Sp = NULL ;

2.3.1.2. *Các phép toán trên Stack*:

Đối với Stack, có hai phép toán chủ yếu là thêm vào (Push) và loại bỏ (Pop)

a. Phép thêm vào (push) : Thêm một phần tử có giá trị x vào đầu Stack.

```
void push(Stack &sp, int x)
{
  Stack p;
  p = (Stack)malloc(sizeof(struct node));
  p->info = x;
  p->next = sp;
  sp = p;
}
```

b. Phép loại bỏ (pop): Xóa phần tử khỏi Stack và trả cho chương trình gọi giá trị của phần tử vừa xóa.

```
int pop(Stack &sp)
{
  Stack p;
  int x;
  if(sp==NULL)
        printf("\nStack rong");
  else
  {
    p = sp;    // nut can xoa la nut dau
```

```
        x =sp->info;
        sp = p->next;
        free(p);
        return x;
      }
   }
```

2.3.2 Queue:

2.3.2.1.*Khái niệm*:

Queue là một danh sách hạn chế mà việc thêm vào được thực hiện ở đầu danh sách, và việc loại bỏ được thực hiện ở đầu còn lại của danh sách (FIFO - First In First Out). Queue chứa các phần tử có cùng kiểu dữ liệu. Ở chương này, ta chỉ tổ chức queue theo danh sách liên kết.

- Vị trí để loại bỏ phần tử được gọi là Front

- Vị trí để thêm vào được gọi là Rear

Queue có hai phép toán chính:

- Insert_queue : thêm một phần tử vào hàng đợi; Trong trường hợp này ta không cần quan tâm hàng đợi bị tràn hay bị đầy.

- Delete_queue: loại bỏ phần tử khỏi Queue và trả về giá trị của phần tử vừa xóa; trước khi xóa, ta phải kiểm tra Queue có khác rỗng hay không.

Lưu ý:

- Phép toán Insert_queue thực hiện ở cuối hàng đợi, còn phép toán Delete_queue thực hiện ở đầu hàng đợi.

- Khai báo: Ta khai báo biến q có kiểu cấu trúc Queue gồm 2 thành phần front, rear là con trỏ chỉ đầu và cuối hàng đợi. Mỗi phần tử của hàng đợi là một nút chứa một số nguyên.

```
struct node
{
  int info;
  struct node *next;
};
typedef node *Hangdoi;
```

```
struct Queue
{
   Hangdoi Front, Rear;
};
struct Queue q;
```

- Khởi tạo hàng đợi: q.Front = NULL;

2.3.2.2. *Các phép toán trên Queue:*

a. Phép thêm vào : Thêm vào cuối danh sách liên kết nên sẽ thay đổi giá trị của Rear

```
void Insert_queue(Queue &q, int x)
{
  Hangdoi p;
  p = (Hangdoi)malloc(sizeof(struct node));
  p->info = x;
  if (q.Front==NULL)
    q.Front=p;
  else q.Rear->next=p;
  q.Rear=p;
  p->next=NULL;
}
```

b. Phép loại bỏ : Loại bỏ phần tử đầu danh sách liên kết, do đó thay đổi giá trị của Front

```
int Delete_queue(Queue &q)
{
  Hangdoi p;
  int x;
  if(q.Front==NULL)
        printf("\nHang doi rong");
  else
  {
```

```
        p = q.Front;    // nut can xoa la nut dau
        x = p->info;
        q.Front = p->next;
        free(p);
        return x;
      }
    }
```

Ví dụ: Viết chương trình đổi số không âm hệ decimal ra số hệ nhị phân, với Stack và Queue là danh sách liên kết đơn.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#define TRUE 1
#define FALSE 0
struct node
{
  int info;
  struct node *next;
};
typedef node *Stack;
Stack sp;
struct node_
{
  int info;
  struct node_ *next;
};
typedef node_ *Hangdoi;
struct Queue
{
```

```
    Hangdoi Front, Rear;
};
struct Queue q;
void push(Stack &sp, int x)
{
  Stack p;
  p = (Stack)malloc(sizeof(struct node));
  p->info = x;
  p->next = sp;
  sp = p;
}
int pop(Stack &sp)
{
  Stack p;
  int x;
  if(sp==NULL)
  {
    printf("\nStack rong");
    getche();
    exit(1);
  }
  else
  {
    p = sp;    // nut can xoa la nut dau
    x =sp->info;
    sp = p->next;
    free(p);
    return x;
  }
}
```

```
void Insert_queue(Queue &q, int x)
{
  Hangdoi p;
  p = (Hangdoi)malloc(sizeof(struct node));
  p->info = x;
  if (q.Front==NULL)
    q.Front=p;
  else q.Rear->next=p;
  q.Rear=p;
  p->next=NULL;
}
int Delete_queue(Queue &q)
{
  Hangdoi p;
  int x;
  if(q.Front==NULL)
  {
    printf("\nHang doi rong");
    getche();
    exit(1);
  }
  else
  {
    p = q.Front;    // nut can xoa la nut dau
    x = p->info;
    q.Front = p->next;
    free(p);
    return x;
  }
}
```

```
void main()
{
  int  sodu;
  float so;
  char c;
  clrscr();
  do
  {
    sp =NULL;  // khoi dong stack
    q.Front =NULL ; // khoi dong hang doi
    printf("\n\nNhap vao mot so thuc khong am: ");
    scanf("%e", &so);
    double positive;
    double r, le, nguyen;
    le = modf(so,&nguyen);
    do
    {
      sodu = (int) nguyen % 2;
      push(sp, sodu);  // push so du vao stack
      nguyen = int(nguyen / 2);
    } while (nguyen != 0);
  printf("So da doi la: ");
  while(sp!=NULL)
     printf("%d", pop(sp));  // pop so du ra khoi stack
 // Doi phan le ra so nhi phan
 if (le!=0)
 {
  printf(".");
  int i=0;
  do
```

```
    {       r =le*2;
            le = modf(r,&positive);
            Insert_queue(q, positive);
            i++ ;
    } while (i <8 && r!=1);
    while (q.Front!=NULL)
    {
      printf("%d", Delete_queue(q));
    }
  }
    printf("\n\nBan co muon tiep tuc khong? (c/k): ");
    c = getche();
  } while(c == 'c' || c == 'C');
}
```

Bài tập

1. Viết chương trình tạo một menu thực hiện các công việc sau:
   a. Nhập danh sách liên kết theo giải thuật thêm về cuối danh sách, mỗi phần tử gồm có các thông tin sau: mssv (int), và hoten (char hoten[30]).
   b. Liệt kê danh sách ra màn hình
   c. Cho biết tổng số nút trong danh sách liên kết, đặt tên hàm là Reccount (int Reccount (NODEPTR First))
   d. Thêm 1 phần tử có nội dung info (mssv, hoten) vào sau phần tử có thứ tự thứ i trong danh sách.

      Ghi chú: - Thứ tự theo qui ước bắt đầu là 1
      - Nếu (i = 0) thêm vào đầu danh sách
      - Nếu i > Reccount(First) thì thêm vào cuối danh sách.

   e. In ra họ tên của sinh viên có mã do ta nhập vào.
   f. Loại bỏ nút có mã do ta nhập vào, trước khi xóa hỏi lại "Bạn thật sự muốn xóa (Y/N) ? "
   g. Sắp xếp lại danh sách theo thứ tự mã số giảm dần.

h. Ghi toàn bộ danh sách vào file tên 'DSSV.DAT'

i. Nạp danh sách từ file 'DSSV.DAT' vào danh sách liên kết. Nếu trong danh sách liên kết đã có nút thì xóa tất cả dữ liệu hiện có trong danh sách liên kết trước khi đưa dữ liệu từ file vào.

2. Viết chương trình tạo một danh sách liên kết theo giải thuật thêm vào đầu danh sách, mỗi nút chứa một số nguyên.
3. Viết hàm tên Delete_Node để xóa nút có địa chỉ p.

   Viết một hàm loại bỏ tất cả các nút có nội dung x trong danh sách liên kết First.
4. Viết hàm Copy_List trên danh sách liên kết để tạo ra một danh sách liên kết mới giống danh sách liên kết cũ.
5. Ghép một danh sách liên kết có địa chỉ đầu là First2 vào một danh sách liên kết có địa chỉ đầu là First1 ngay sau phần tử thứ i trong danh sách liên kết First1.
6. Viết hàm lọc danh sách liên kết để tránh trường hợp các nút trong danh sách liên kết bị trùng info.
7. Đảo ngược vùng liên kết của một danh sách liên kết sao cho:
   - First sẽ chỉ đến phần tử cuối
   - Phần tử đầu có liên kết là NULL.
8. Viết hàm Left_Traverse (NODEPTR First) để duyệt ngược danh sách liên kết.
9. Viết giải thuật tách một danh sách liên kết thành hai danh sách liên kết, trong đó một danh sách liên kết chứa các phần tử có số thứ tự lẽ và một danh sách liên kết chứa các phần tử có số thứ tự chẵn trong danh sách liên kết cũ.
10. Tạo một danh sách liên kết chứa tên học viên, điểm trung bình, hạng của học viên (với điều kiện chỉ nhập tên và điểm trung bình). Quá trình nhập sẽ dừng lại khi tên nhập vào là rỗng.

    Xếp hạng cho các học viên. In ra danh sách học viên thứ tự hạng tăng dần (Ghi chú : Cùng điểm trung bình thì cùng hạng).
11. Nhập hai đa thức theo danh sách liên kết. In ra tích của hai đa thức này.

    Ví dụ: Đa thức First1 : $2x^5+4x^2-1$

    Đa thức First2 : $10x^7-3x^4+x^2$

$\Rightarrow$ Kết quả in ra : $20x^{12} + 34x^9 - 8x^7 - 12x^6 + 7x^4 - x^2$

(Ghi chú : Không nhập và in ra các số hạng có hệ số bằng 0)

12. Viết giải thuật thêm phần tử có nội dung x vào danh sách liên kết có thứ tự tăng dần sao cho sau khi thêm danh sách liên kết vẫn có thứ tự tăng.
13. Loại bỏ phần tử có nội dung là x trong danh sách liên kết có thứ tự tăng dần.
14. Cho 2 danh sách liên kết First1, First2 có thứ tự tăng dần theo info. Viết giải thuật Merge để trộn 2 danh sách liên kết này lại sao cho danh sách liên kết sau khi trộn cũng có thứ tự tăng dần.

Bài tập nghiên cứu thêm

Bài tập 1: Hãy chuyển một số nguyên dương sang cơ số bất kỳ (sử dụng ngăn xếp)

Bài tập 2: Dùng hàng đợi và ngăn xếp để viết hàm thực hiện đảo ngược các phần tử trong hàng đợi.

Bài tập 3: Hãy viết chương trình quản lý phòng trong khách sạn dùng hàng đợi.

Bài tập 4: Hãy viết chương trình thực hiện lọc các ký tự nhập vào từ bàn phím, chỉ giữ lại các ký tự số.

CHƯƠNG 4: SẮP XẾP

Mã bài: MH28.4

Mục tiêu:

- Xác định được tính chất của việc sắp xếp dữ liệu;
- Nêu được ý tưởng, thuật toán chi tiết của một số phương pháp sắp xếp;
- Áp dụng một số thuật toán sắp xếp vào thực hiện việc sắp xếp các dãy khóa cụ thể;
- Cài đặt được các thuật toán sắp xếp trong ngôn ngữ lập trình bậc cao;
- Nghiêm túc, tỉ mỉ, sáng tạo trong việc học và vận dụng vào làm bài tập.

2.1. Sắp xếp kiểu chọn, chèn, nổi bọt

2.1.1 Phương pháp sắp xếp lựa chọn

Ý tưởng chính của phương pháp sắp xếp đơn giản: "Ở lượt thứ i (i=0, 1, …, n-2) chúng ta sẽ chọn trong ai, ai+1, …, an-1 phần tử nhỏ nhất và đổi chỗ nó với ai". Hay nói cách khác ở lượt thứ i chúng ta sẽ chọn phần tử a[j] (j=i+1, …, n-1) là phần tử nhỏ nhất trong các phần tử a[j], a[j+1], …, a[n-1].

Như vậy, sau j lượt, j khoá nhỏ hơn sẽ lần lượt ở vị trí thứ nhất, thứ hai, …, thứ j theo đúng thứ tự sắp xếp.

Sau đây là giải thuật:

Sắp xếp tăng dần:

```
void selection_sort(int a[], int n)
{
 int i,j; //hai bien chay
 int tg,m; //hai bien trung gian
 //bien tg dung de bien doi
 //bien m dung de xac dinh so nho nhat
 for (i=0;i<=n-2;i++)
 {
  m = i;
    //chon phan tu nho nhat
  for (j=i+1;j<=n-1;j++)
   if (a[j]<a[m])
    m = j;
    //neu phan tu nho nhat khac i
    //thi hoan doi
  if (m!=i)
   {
    tg = a[m];
    a[m] = a[i];
    a[i] = tg;
   }
  }
}
```

Sắp xếp giảm dần:

```
void selection_sort(int a[], int n)
{
 int i,j; //hai bien chay
 int tg,m; //hai bien trung gian
```

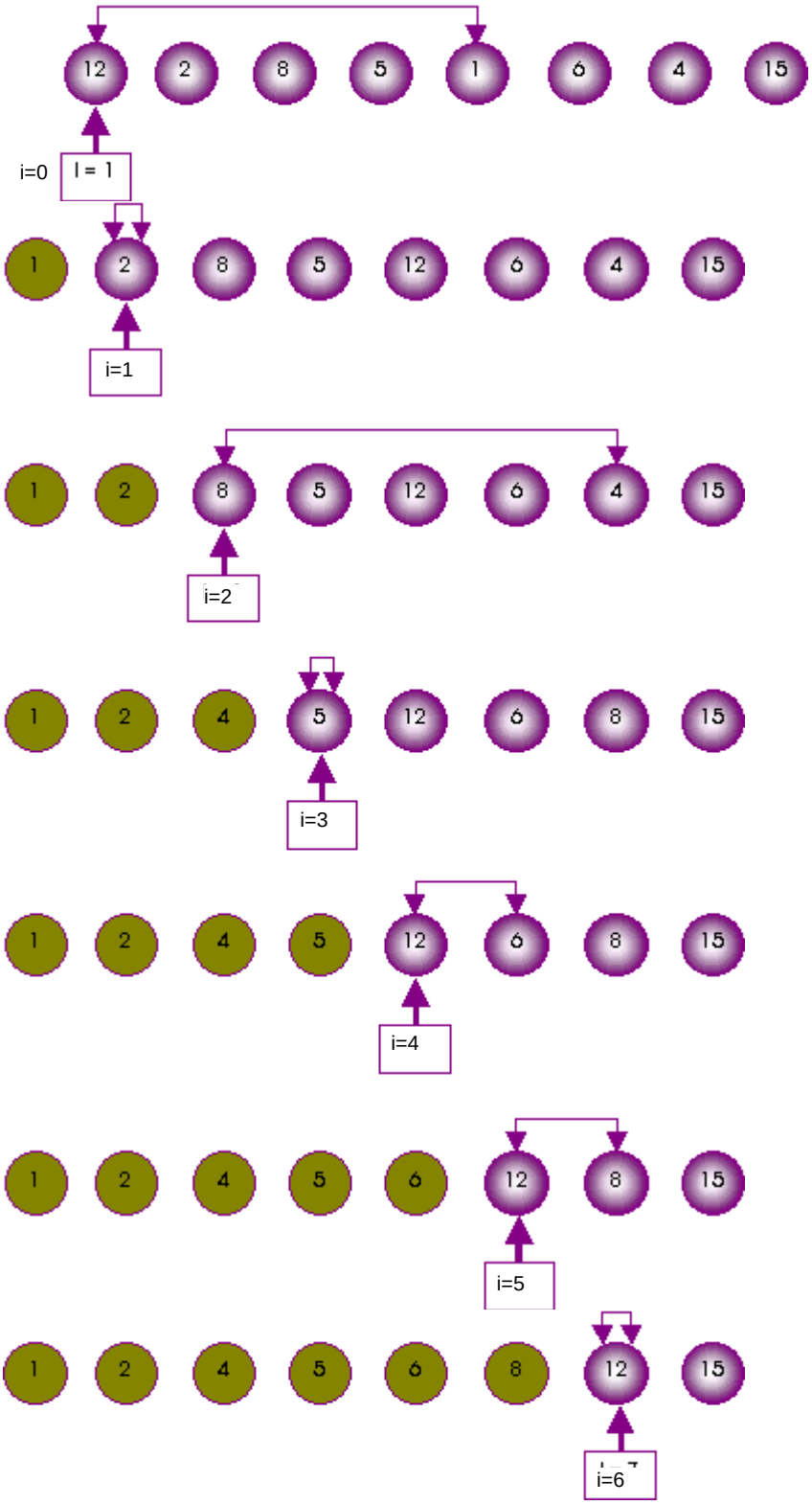
```
//bien tg dung de bien doi
//bien m dung de xac dinh so lon nhat
for (i=0;i<=n-2;i++)
{
 m = i;
   //chon phan tu lon nhat
 for (j=i+1;j<=n-1;j++)
  if (a[j]>a[m])
   m = j;
   //neu phan tu lon nhat khac i
   //thi hoan doi
 if (m!=i)
  {
   tg = a[m];
   a[m] = a[i];
   a[i] = tg;
  }
 }
}
```

Ví dụ 1: Sắp xếp mảng gồm 10 số nguyên: 5, 6, 2, 2, 10, 12, 9, 10, 9 và 3

Mảng / Bước	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Ban đầu	5	6	2	2	10	12	9	10	9	3
Bước 1	2	6	5	2	10	12	9	10	9	3
Bước 2		2	5	6	10	12	9	10	9	3
Bước 3			3	6	10	12	9	10	9	5

Bước 4				5	10	12	9	10	9	6
Bước 5					6	12	9	10	9	10
Bước 6						9	12	10	9	10
Bước 7							9	10	12	10
Bước 8								10	12	10
Bước 9									10	12
Kết quả	2	2	3	5	6	9	9	10	10	12

Ví dụ 2: Minh họa sắp xếp dãy 12 2 8 5 1 6 4 15



Ví dụ 3: Chương trình minh họa thuật toán sắp xếp lựa chọn.

```
//chuong trinh minh ho thuat toan sap xep lua chon
#include <stdio.h>
```

```
#include <stdlib.h>
#include <conio.h>
#include<iostream.h>
#define MAXLIST 200
#define TRUE 1
#define FALSE 0

int nodes[MAXLIST];
//Sap xep giam dan:
void selection_sort(int a[], int n)
{
 int i,j; //hai bien chay
 int tg,m; //hai bien trung gian
 //bien tg dung de bien doi
 //bien m dung de xac dinh so lon nhat
 for (i=0;i<=n-2;i++)
 {
  m = i;
    //chon phan tu lon nhat
  for (j=i+1;j<=n-1;j++)
   if (a[j]>a[m])
    m = j;
    //neu phan tu lon nhat khac i
    //thi hoan doi
  if (m!=i)
   {
    tg = a[m];
    a[m] = a[i];
    a[i] = tg;
   }
```

```
  }
}
// Chuong trinh chinh
void main()
{
  int chucnang;
  char c;

  clrscr();

  do
  {
    // menu chinh cua chuong trinh
    cout<<"\n\n\tSELECTION SORT";
    cout<<"\n\nCac chuc nang cua chuong trinh:\n";
    cout<<"   1: Tao danh sach voi"<<MAXLIST<<"so ngau nhien\n";
    cout<<"   2: Seleciton sort giam dan\n";
    cout<<"   3: Duyet danh sach\n";
    cout<<"   0: Ket thuc chuong trinh\n";
    cout<<"Chuc nang ban chon: ";
    cin>>chucnang;
    switch(chucnang)
    {
       case 1:
       {
          for(int i = 0; i < MAXLIST; i++)
            nodes[i] = random(1000);
          cout<<"\nDa tao xong danh sach voi"<<MAXLIST<<"so ngau nhien";
          break;
       }
```

```
            case 2:
            {
              selection_sort(nodes, MAXLIST);
              cout<<"\nDa sap xep xong danh sach theo giai thuat selection sort giam dan";
              break;
            }
            case 3:
            {
              cout<<"\nDUYET DANH SACH:\n";
              for(int i = 0; i < MAXLIST; i++)
                cout<<nodes[i]<<"  ";
              break;
            }
          }
        } while(chucnang != 0);
    }
```

2.1.2. Sắp xếp kiểu chèn

Ý tưởng chính của thuật toán như sau: Giả sử đã có i-1 số xếp theo thứ tự, muốn thêm vào một số thứ i thì chúng ta sẽ lần lượt so sánh các vị trí từ i-1, i-2, … để chọn vị trí thích hợp chèn số này vào.

Chúng ta thiết kế thuật toán như sau: Giả sử đã có thứ tự sắp xếp từ a[0], a[1], …,a[i-1], chúng ta xét vị trí để chèn số a[i] vào cho thích hợp. Thuật toán được thiết kế như sau:

Sắp xếp tăng dần:

```
void insertion_sort(int a[], int n)
{
 int x;
 int i,j;
 for (i=1;i<=n-1;i++)
 {
```

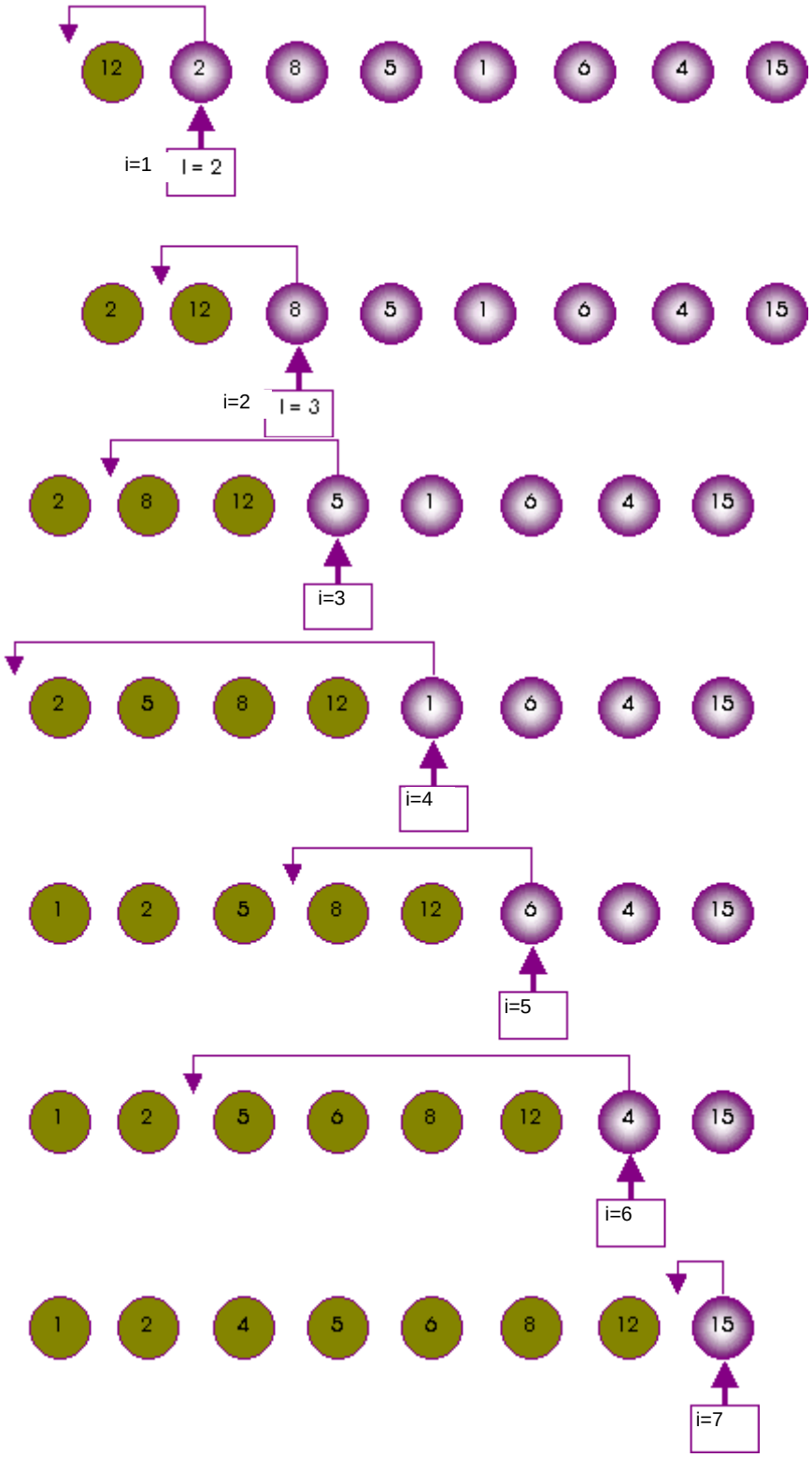
```
  x = a[i]; j = i-1; // x là phần tử a[i] đang xét
  while (j>=0 && x<a[j])
  {
   a[j+1] = a[j]; //vừa xét và tìm vị trí thích hợp
   j--; // vừa thực hiện việc đẩy lùi danh sách (chèn)
  }
  a[j+1] = x; //chèn vào vị trí thích hợp
 }
}
```

Sắp xếp giảm dần:

```
void insertion_sort(int a[], int n)
{
 int x;
 int i,j;
 for (i=1;i<=n-1;i++)
 {
  x = a[i]; j = i-1; // x là phần tử a[i] đang xét
  while (j>=0 && x>a[j])
  {
   a[j+1] = a[j]; //vừa xét và tìm vị trí thích hợp
   j--; // vừa thực hiện việc đẩy lùi danh sách (chèn)
  }
  a[j+1] = x; //chèn vào vị trí thích hợp
 }
}
```

Ở thuật toán trên, chúng ta chú ý việc đẩy lùi các số để tạo ra vị trí chèn (a[j+1]) được thực hiện song song với việc tìm kiếm vị trí chèn thích hợp (a[j+1]).

Ví dụ 4: Minh họa sắp xếp dãy số 12 ; 2; 8; 5; 1; 6; 4;15



2.1.3 Sắp xếp kiểu nổi bọt

- Nội dung : Cho dãy A có n phần tử. Ta cho i duyệt dãy a[0],...,a[n-1]; nếu a[i-1] lớn hơn a[i] thì ta hoán đổi (a[i-1],a[i]). Lặp lại quá trình duyệt dãy này cho đến khi không có xảy ra việc đổi chỗ của hai phần tử.

Ví dụ 5: Ta sắp thứ tự dãy số sau : 26 33 35 29 19 12 32

Bước 0	**1**	**2**	**3**	**4**	**5**	**6**
26	12	12	12	12	12	12
33	26	19	19	19	19	19
35	33	26	26	26	26	26
29	35	33	29	29	29	29
19	29	35	33	32	32	32
12	19	29	35	33	33	33
32	32	32	32	35	35	35

Minh họa quá trình sắp xếp qua phương pháp Buble Sort

- Chương trình:

```
 void Bubble_Sort(int A[100], int n)
 { int i,j,temp;
  for (i=1; i<n; i++)
    for (j=n-1;j>=i; j--)
      if (A[j-1] > A[j])
       {  temp = A[j-1];
 A[j-1] = A[j];
 A[j] = temp;
       }
 }
```

Ta nhận thấy phương pháp này có thể được cải tiến dễ dàng. Nếu ở lần duyệt dãy nào đó mà không có có sự đổi chỗ giữa hai phần tử thì dãy đã có thứ tự và giải thuật kết thúc. Trong trường hợp này, ta dùng một cờ hiệu flag để ghi nhận điều này, và giải thuật Bubble Sort được cải tiến như sau:

```
#define FALSE 0
#define TRUE  1
    void Bubble_Sort_Ad(int A[], int n)
{ int i,temp;
```

```
    unsigned char flag=TRUE;
    while (flag)
    { flag = FALSE ;
for (i=0; i<n-1; i++)
      if (A[i] > A[i+1])
        {   temp = A[i];
            A[i] = A[i+1];
            A[i+1] = temp;
            flag=TRUE;
        }
    }
  }
```

2.2. Sắp xếp bằng phương pháp trộn

Để sắp xếp dãy a0, a1, ..., an-1, giải thuật Merge Sort dựa trên nhận xét sau:

Mỗi dãy a0, a1, ..., an-1 bất kỳ đều có thể coi như là một tập hợp các dãy con liên tiếp mà mỗi dãy con đều đã có thứ tự. Ví dụ dãy 12, 2, 8, 5, 1, 6, 4, 15 có thể coi như gồm 5 dãy con không giảm (12); (2, 8); (5); (1, 6); (4, 15).

Dãy đã có thứ tự coi như có 1 dãy con.

Như vậy, một cách tiếp cận để sắp xếp dãy là tìm cách làm giảm số dãy con không giảm của nó. Đây chính là hướng tiếp cận của thuật toán sắp xếp theo phương pháp trộn.

Trong phương pháp Merge sort, mấu chốt của vấn đề là cách phân hoạch dãy ban đầu thành các dãy con. Sau khi phân hoạch xong, dãy ban đầu sẽ được tách ra thành 2 dãy phụ theo nguyên tắc phân phối đều luân phiên. Trộn từng cặp dãy con của hai dãy phụ thành một dãy con của dãy ban đầu, chúng ta sẽ nhân lại dãy ban đầu nhưng với số lượng dãy con ít nhất giảm đi một nửa. Lặp lại qui trình trên sau một số bước, chúng ta sẽ nhận được 1 dãy chỉ gồm 1 dãy con không giảm. Nghĩa là dãy ban đầu đã được sắp xếp.

4.2.4.1. Phương pháp trộn trực tiếp

Giải thuật trộn trực tiếp là phương pháp trộn đơn giản nhất. Việc phân hoạch thành các dãy con đơn giản chỉ là tách dãy gồm n phần tử thành n dãy con. Đòi hỏi

của thuật toán về tính có thứ tự của các dãy con luôn được thỏa trong cách phân hoạch này vì dãy gồm một phân tử luôn có thứ tự. Cứ mỗi lần tách rồi trộn, chiều dài của các dãy con sẽ được nhân đôi.

Các bước thực hiện thuật toán như sau:

Bước 1 : // Chuẩn bị

k = 1; // k là chiều dài của dãy con trong bước hiện hành

Bước 2 :

Tách dãy a0, a1, ., an-1 thành 2 dãy b, c theo nguyên tắc luân phiên từng nhóm k phần tử:

b = a0, ., ak, a2k+1, ., a3k, .

c = ak+1, ., a2k, a3k+1, ., a4k, .

Bước 3 :

Trộn từng cặp dãy con gồm k phần tử của 2 dãy b, c vào a.

Bước 4 :

k = k*2;

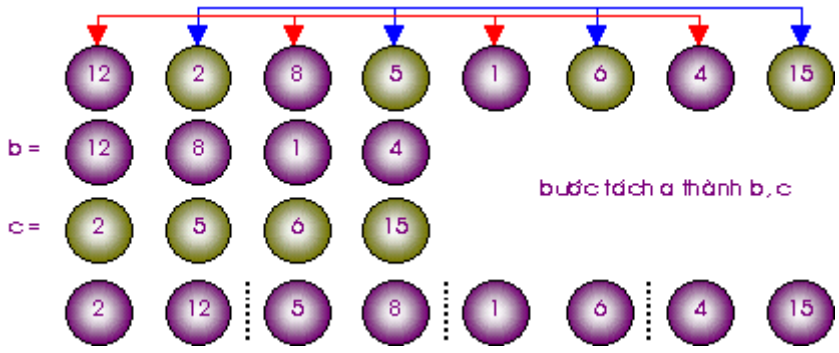
Nếu k < n thì trở lại bước 2.

Ngược lại: Dừng

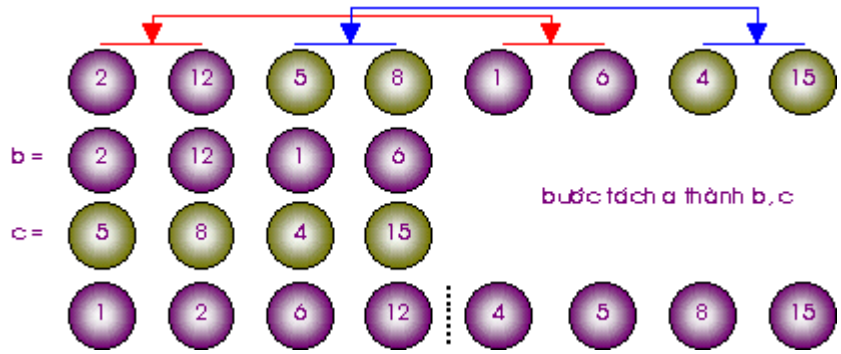
Ví dụ Cho dãy số a:

12 2 8 5 1 6 4 15

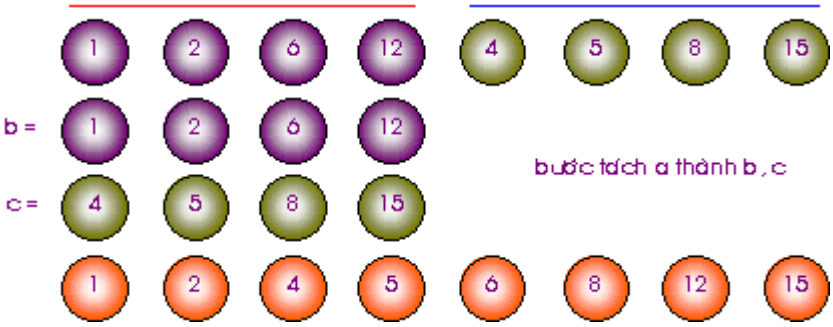
k = 1:



k = 2:



k = 4:



Cài đặt

```
int     b[MAX], c[MAX]; // hai mảng phụ
void MergeSort(int a[], int n)
{
   int   p, pb, pc;   // các chỉ số trên các mảng a, b, c
   int   i, k = 1;    // độ dài của dãy con khi phân hoạch
   do    {
   // tách a thanh b và c;
   p = pb = pc = 0;
   while(p < n) {
   for(i = 0; (p < n)&&(i < k); i++)
         b[pb++] = a[p++];
   for(i = 0; (p < n)&&(i < k); i++)
         c[pc++] = a[p++];
   }
   Merge(a, pb, pc, k); //trộn b, c lại thành a
k *= 2;
```

```
    }while(k < n);
}
```

Trong đó hàm Merge có thể được cài đặt như sau :

```
void Merge(int a[], int nb, int nc, int k)
{  int p, pb, pc, ib, ic, kb, kc;
p = pb = pc = 0; ib = ic = 0;
while((0 < nb)&&(0 < nc))
{
  kb = min(k, nb); kc = min(k, nc);
  if(b[pb+ib] <= c[pc+ic])
   {
    a[p++] = b[pb+ib]; ib++;
     if(ib == kb)
     {
        for(; ic<kc; ic++) a[p++] = c[pc+ic];
        pb += kb; pc += kc; ib = ic = 0;
       nb -= kb; nc -= kc;
     }
  }
   else
    {
      a[p++] = c[pc+ic]; ic++;
        if(ic == kc)
        {
           for(; ib<kb; ib++) a[p++] = b[pb+ib];
           pb += kb; pc += kc; ib = ic = 0;
         nb -= kb; nc -= kc;
         }
     }
 }
```

}

2.3. Sắp xếp nhanh (Quick sort)

Ý tưởng chính của phương pháp này có thể tóm tắt như sau:

Chọn một khoá nào đó của dãy làm "chốt", các phần tử nhỏ hơn chốt phải được sắp vào phía trước "chốt"; những phần tử ngược lại phải được sắp bên phải "chốt". Muốn vậy, các phần tử của dãy sẽ được so sánh với "chốt" và sẽ đổi vị trí cho nhau.

Chia dãy cần sắp xếp thành hai phần, lấy giá trị ở giữa X làm chuẩn để so sánh.

Đi tìm phần tử A ở dãy trên có giá trị lớn hơn X.

Đi tìm phần tử B ở dãy dưới có giá trị nhỏ hơn X.

Hoán vị 2 phần tử A, B.

Tiếp tục như vậy cho đến khi chúng ta đạt được dãy trên chứa các giá trị nhỏ hơn X, dãy dướI chứa các giá trị lớn hơn X.

Tiếp tục áp dụng thuật toán trên vào hai dãy con trên và dưới (đệ quy) cho đến khi không chia được nữa thì việc sắp xếp hoàn tất.

Cụ thể là xét một đoạn của dãy từ thành phần thứ L đến thành phần thứ R chúng ta thực hiện các thao tác:

* Lấy giá trị của thành phần thứ (L+R)/2 gán vào biến X.

* Cho i ban đầu bằng L.

* Cho j ban đầu bằng R.

* Lặp lại:

Chừng nào còn a[i]<x thì tăng i.

Chừng nào a[j] > x thì giảm j.

Nếu i<=j thì

- Hoán vị a[i] và a[j]

- Tăng i.

- Giảm j.

Cho đến khi i>j

* Sắp xếp đoạn từ a[l] đến a[j].

* Sắp xếp đoạn từ a[i] đến a[r].

Thuật toán được viết như sau:

Sắp xếp tăng dần:

```
void quicksort(int a[], int l, int r)
{
   int i,j;
   int tg,x;
   i = l;
   j = r;
   x = a[(l+r)/2];
 do
 {
  while (a[i]<x)
    i++;
  while (a[j]>x)
    j--;
   if (i<=j)
   {
     tg = a[i];
     a[i] = a[j];
     a[j] = tg;
     i++;j--;
   }
 } while(i<=j);
if (l<j)    // phan thu nhat co tu 2 phan tu tro len
    quicksort(a,l,j);
if (i<r)    // phan thu 2 co tu 2 phan tu tro len
    quicksort (a,i,r);
}
```

Sắp xếp giảm dần:

```
void quicksort(int a[], int l, int r)
{
   int i,j;
```

```
    int tg,x;
    i = l;
    j = r;

    x = a[(l+r)/2];
  do
  {
   while (a[i]>x)
    i++;
   while (a[j]<x)
    j--;
   if (i<=j)
   {
     tg = a[i];
     a[i] = a[j];
     a[j] = tg;
      i++;j--;
      }
  }
  while (i<=j);
  if (l<j)    // phan thu nhat co tu 2 phan tu tro len
     quicksort(a,l,j);
  if (i<r)    // phan thu 2 co tu 2 phan tu tro len
     quicksort (a,i,r);
}
```

Bài Tập:

Bài tập 1: Thiết kế lại tất cả các phương pháp sắp xếp sắp xếp danh sách là chuỗi ký tự.

Bài tập 2: Thiết kế lại phương pháp tìm kiếm nhị phân để tìm kiếm trên dãy sắp xếp giảm dần.

Bài tập 3: Viết chương trình nhập vào một dãy các phần tử là chuỗi, sắp xếp dãy này theo thứ tự tăng dần và in kết quả ra màn hình.

Bài tập 4: Viết chương trình nhập vào một dãy họ tên học viên. Sau đó sắp xếp tăng dần và tìm kiếm xem họ tên một học viên có tồn tại trong danh sách hay không.

Bài tập 5: Viết thuật toán sắp xếp cho danh sách liên kết.

CHƯƠNG 5: TÌM KIẾM

Mã bài: MH28.5

Mục tiêu:

- Nêu được ý tưởng, thuật toán chi tiết của một số phương pháp tìm kiếm;
- Áp dụng một số thuật toán tìm kiếm vào các dãy khóa cụ thể;
- Cài đặt được các thuật toán tìm kiếm trong ngôn ngữ lập trình bậc cao;
- Nghiêm túc, tỉ mỉ, sáng tạo trong việc học và vận dụng vào làm bài tập.

5.1 Tìm kiếm tuần tự

Tìm kiếm tuần tự là kỹ thuật tìm kiếm đơn giản và cổ điển. Ý tưởng:

"Bắt đầu từ bản ghi đầu tiên, lần lượt so sánh khoá tìm kiếm với khoá tương ứng của các bản ghi trong bảng, cho tới khi tìm được bản ghi mong muốn hoặc đã hết bảng ghi mà vẫn chưa tìm thấy"

Mã lệnh hàm như sau:

```
int  find(int a[], int n, int x)
{
  int kt = 0;
  int i;
  for (i=0;i<=n-1;i++)
   if (a[i]==x)
    {
      kt = i;
      break;'
   }
 return kt;
}
```

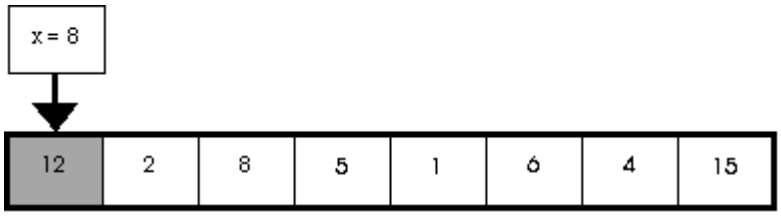
Ví dụ 1: Minh họa cách thực hiện của chương trình tìm kiếm tuyến tính.

Cho dãy số a:

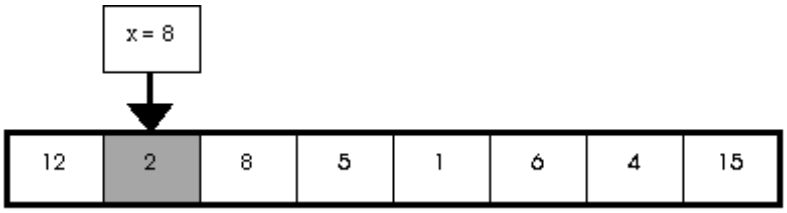
12 2 8 5 1 6 4 15

Nếu giá trị cần tìm là x=8, giải thuật được tiến hành như sau :

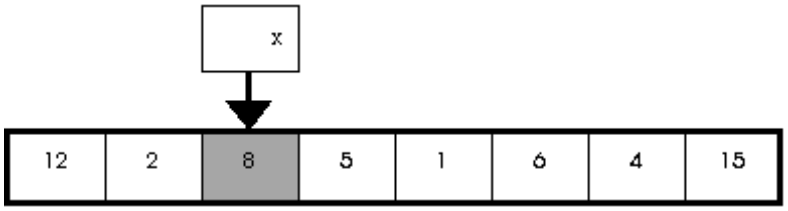
i = 0



i=1



i=2: Dừng



5.2 Tìm kiếm nhị phân

Phương pháp này yêu cầu dãy phải được sắp xếp tăng dần hoặc giảm dần. Thuật toán áp dụng cho dãy a sắp xếp tăng dần như sau:

Phần tử đầu của dãy có chỉ số l.

Phần tử cuối của dãy có chỉ số r.

Lặp lại:

- m=(l+r)/2
- nếu a[m]<x thì r=m-1
- nếu a[m]=x thì đã tìm thấy, dừng ngay thuật toán.
- nếu a[m]>x thì l=r+1

cho đến khi l>r.

Thuật toán thể hiện:

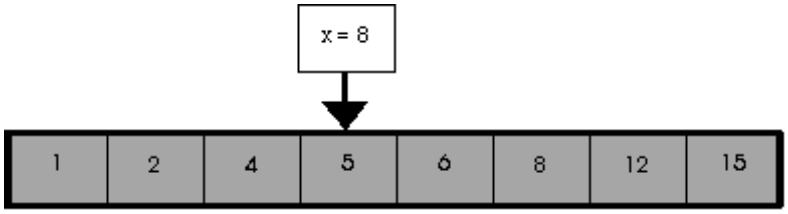
```
int  BinarySearch(int a[], int n, int x)
{
   int kt = 0;
   int l,r,m;
```

```
 while (l<=r)
{
   m = (l+r)/2;
   if  (x<a[m])
     r= m– 1;
   else
    if (x==a[m])
      {
         kt = 1;
        break;
        }
     else
         l =  m +1;
 }
return kt;
}
```

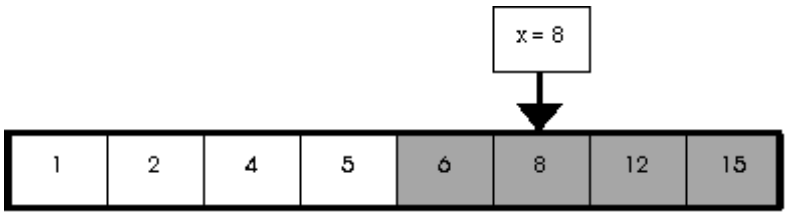
Ví dụ 2: Cho dãy số a gồm 8 phần tử: 1; 2;4;5;6;8;12;15

Nếu giá trị cần tìm là 8, giải thuật được tiến hành như sau:

l = 0, r = 7, m=3



left = 4, right = 7, midle = 5



Dừng.

Bài tập

Bài tập 1. Cài đặt các thuật toán tìm kiếm đã trình bày. Thể hiện trực quan các thao tác của thuật toán. Tính thời gian thực hiện của mỗi thuật toán.

Bài tập 2. Hãy viết hàm tìm tất cả các số nguyên tố nằm trong mảng một chiều a có n phần tử.

Bài tập 3.Hãy viết hàm tìm dãy con tăng dài nhất của mảng một chiều a có n phần tử (dãy con là một dãy liên tiếp các phần của a).

Bài tập 4: Hãy viết hàm tìm kiếm một phần tử trong danh sách liên kết.

TÀI LIỆU THAM KHẢO

[1] Đỗ Xuân Lôi (1999), *Cấu trúc dữ liệu và giải thuật*, NXB Thống kê.
[2] Hoàng Nghĩa Tý (2000), *Cấu trúc dữ liệu và thuật toán*, NXB Xây dựng.