TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Lời nói đầu

250 bài tập kỹ thuật lập trình C (230 bài tập chính thức, 20 bài tập bổ sung) trong tập sách này được chọn lọc từ các bài tập thực hành môn Ngôn ngữ lập trình C và Lập trình Cấu trúc dữ liệu bằng ngôn ngữ C cho sinh viên Đại học và Cao đẳng chuyên ngành Công nghệ Thông tin.

Các bài tập đã được sắp xếp theo một trình tự nhất định, nhằm đảm bảo cho người đọc nắm vững một cách có hệ thống các kiến thức cần thiết của kỹ thuật lập trình nói chung và ngôn ngữ lập trình C nói riêng; chuẩn bị nền tảng cho các môn học có liên quan. Mặc dù cố gắng duyệt qua các vấn đề cơ bản của ngôn ngữ lập trình C, nhưng tập sách này được viết với mục tiêu củng cố và nâng cao khả năng làm việc với ngôn ngữ C.

Khác với các sách bài tập khác, các bài tập trong tập sách này đều có hướng dẫn giải chi tiết. Khi hướng dẫn giải bài tập, chúng tôi cố gắng:

- Thể hiện một góc nhìn riêng về kỹ thuật lập trình bằng ngôn ngữ C, chú ý đến những đặc điểm của ngôn ngữ C. Nói cách khác, chúng tôi chú ý đến lập trình theo phong cách của C.
- Phân tích quá trình tư duy khi giải quyết vấn đề, củng cố các kiến thức toán học cũng như lập trình cơ bản, nhằm làm nổi bật vai trò của ngôn ngữ lập trình như một công cụ hỗ trợ mang tính thực tế cao.
- Lập trình thật ngắn gọn và rõ ràng giúp người đọc hiểu rõ vấn đề. Nâng cao kỹ năng lập trình. Người đọc sẽ thấy thú vị và bất ngờ với một số kỹ thuật giải quyết vấn đề.
- Theo chuẩn ANSI/ISO C89 phù hợp với nhà trường ở Việt nam, chuẩn mới nhất là ANSI/ISO C11 (ISO/IEC 9899:2011).
- Các bài giải của 250 bài tập và các phương án giải khác đã được kiểm tra bằng Cppcheck 1.72 (cppcheck.sourceforge.net).

Chúng tôi tin rằng tập sách này sẽ giúp người đọc thật sự củng cố và nâng cao kiến thức lập trình với ngôn ngữ C.

Mặc dù đã dành rất nhiều thời gian và công sức cho tập sách, phải hiệu chỉnh nhiều lần và chi tiết, nhưng tập sách không thể nào tránh được những sai sót và hạn chế. Chúng tôi thật sự mong nhận được các ý kiến góp ý từ bạn đọc để tập sách có thể hoàn thiện hơn.

Xin chân thành cảm ơn anh Lê Gia Minh đã xem và đóng góp nhiều ý kiến quý giá cho tập sách. Cảm ơn bạn Nguyễn Đình Song Toàn đã khuyến khích tôi học C. Cảm ơn các anh Thân Văn Sử, Lê Mậu Long, Nguyễn Minh Nam, tôi đã học tập được rất nhiều kinh nghiệm từ các anh.

Phiên bản

Cập nhật ngày: 08/03/2016

Thông tin liên lạc

Mọi ý kiến và câu hỏi có liên quan xin vui lòng gửi về:

Dương Thiên Tứ
91/29 Trần Tấn, P. Tân Sơn Nhì, Q. Tân Phú, Thành phố Hồ Chí Minh
Facebook: https://www.facebook.com/tu.duongthien
E-mail: thientu2000@yahoo.com

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Hướng dẫn sử dụng tài liệu

Trong giáo trình thực hành này, các bạn sẽ thực hiện các bài tập lập trình cơ bản, được thực hiện bằng ngôn ngữ lập trình C, theo chuẩn ANSI/ISO C89 (ANS X3. 159-1989 và ISO/IEC 9899 - 1990).

ANSI/ISO C99 (ISO/IEC 9899 - 1999) hiện chưa dùng phổ biến tại nhà trường ở Việt nam, bạn có thể tham khảo thêm từ các tài liệu giới thiệu trong phần tham khảo.

Hướng dẫn thực hiện bài tập thực hành

- Các bạn nên thực hiện toàn bộ các bài tập thực hành. Các bài tập này đã được tuyển chọn và sắp xếp để mang đến cho các bạn kiến thức cơ bản và tổng quát về ngôn ngữ lập trình C. Các bạn nên:

✍ Đọc kỹ bài tập để hiểu rõ yêu cầu bài tập.

✍ Dành nhiều thời gian thiết kế cẩn thận chương trình. Nhiều vấn đề lập trình sẽ nảy sinh do thiết kế sai, và nếu bạn mất nhiều thời gian để thiết kế bạn sẽ rút ngắn được giai đoạn viết code và dò lỗi. Luôn luôn thử tìm một cách đơn giản nhất để thiết kế chương trình.

- Nếu chương trình có lỗi và không chạy được, trước khi xem bài giải, hãy chắc rằng bạn đã:

✍ Mất nhiều thời gian để cố gắng giải bài tập theo cách của bạn;

✍ Thử dùng tiện ích dò lỗi (debugger) nếu chương trình có lỗi;

✍ Đọc kỹ lại bài học lý thuyết có liên quan;

✍ Thử mọi cách mà bạn nghĩ có thể giải được bài tập.

- Một số chi tiết:

✍ Các chương trình không yêu cầu kiểm tra chặt chẽ dữ liệu nhập. Tuy nhiên, có thể dùng hàm `assert()` để kiểm tra các tiền điều kiện (pre-condition).

✍ Các bài tập có thể thực hiện hai phiên bản: giải quyết vấn đề trực tiếp trong hàm `main()`, hoặc viết các hàm phụ để giải quyết từng vấn đề riêng tùy theo yêu cầu và độ phức tạp của bài tập (hàm `main()` xem như một test driver).

✍ Các bài tập về mảng (array) và chuỗi (string) thực hiện hai phiên bản: không dùng con trỏ và dùng con trỏ (cấp phát động).

- Xem bài giải:

Bài giải chỉ trình bày một trong các lời giải có thể có của bài tập. Chúng tôi đã cố đa dạng hóa cách giải để bạn có thể rút được nhiều kiến thức và kinh nghiệm từ bài giải. Bạn cũng có thể học tập thêm cách tiếp cận vấn đề, cách viết code, …

Bạn chỉ xem bài giải khi đã thực hiện xong bài tập, so sánh với bài giải của bạn để có thêm kinh nghiệm.

Ghi chú dùng trong sách

Thông tin, kiến thức hỗ trợ cần có để thực hiện bài tập.

Ví dụ xuất mẫu của chương trình. Dùng để kiểm tra nhanh chương trình.

Gợi ý giải bài tập.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

KHÁI NIỆM CƠ BẢN - TOÁN TỬ
CẤU TRÚC LỰA CHỌN - CẤU TRÚC LẶP

Bài 1: Nhập vào diện tích S của một mặt cầu. Tính thể tích V của hình cầu này.

$$\begin{cases} S = 4\pi R^2 \\ V = \dfrac{4}{3}\pi R^3 \end{cases} \quad (\pi \approx 3.141593)$$

```
Nhap dien tich S: 256.128 ↵
The tich V = 385.442302
```

Bài giải: xem trang 66

Bài 2: Nhập vào tọa độ 2 điểm A(xA, yA) và B(xB, yB). Tính khoảng cách AB.

$$|AB| = \sqrt{(xB - xA)^2 + (yB - yA)^2}$$

```
A(xA, yA)? 3.2 -1.4 ↵
B(xB, yB)? -5.7 6.1 ↵
|AB| = 11.6387
```

Bài giải: xem trang 66

Bài 3: Viết chương trình nhập vào tọa độ (xC, yC) là tâm của một đường tròn, và R là bán kính của đường tròn đó. Nhập vào tọa độ (xM, yM) của điểm M. Xác định điểm M nằm trong, trên hay nằm ngoài đường tròn.

```
Nhap toa do tam C(xC, yC)? 0.5 4.3 ↵
Nhap ban kinh R? 7.4 ↵
Nhap toa do M(xM, yM)? 3.2 6.5 ↵
M nam trong C()
```

Bài giải: xem trang 66

Bài 4: Viết chương trình nhập vào ba số thực là ba cạnh của một tam giác. Kiểm tra ba cạnh được nhập có hợp lệ hay không. Nếu hợp lệ, hãy cho biết loại tam giác và tính diện tích tam giác đó.

Tổng hai cạnh bất kỳ của một tam giác phải lớn hơn cạnh còn lại.
Công thức Heron[1] dùng tính diện tích tam giác theo chu vi:
$S = \sqrt{p(p-a)(p-b)(p-c)}$, trong đó p là nửa chu vi: $p = \dfrac{a+b+c}{2}$

```
Nhap 3 canh tam giac: 3 4 5 ↵
Tam giac vuong
Dien tích S = 6
```

Bài giải: xem trang 67

Bài 5: Viết chương trình nhập vào tọa độ các đỉnh của tam giác ABC và của điểm M. xác định điểm M nằm trong, nằm trên cạnh hay nằm ngoài tam giác ABC.

[1] Heron of Alexandria (10 - 70)

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Công thức tính diện tích một tam giác theo tọa độ 3 đỉnh của nó:

$$S_{ABC} = \frac{1}{2}\begin{Vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_C & y_C & 1 \end{Vmatrix} = \frac{1}{2}\left| x_A\begin{vmatrix} y_B & 1 \\ y_C & 1 \end{vmatrix} - y_A\begin{vmatrix} x_B & 1 \\ x_C & 1 \end{vmatrix} + \begin{vmatrix} x_B & y_B \\ x_C & y_C \end{vmatrix} \right|$$

$$= \frac{1}{2}\left|x_A(y_B - y_C) - y_A(x_B - x_C) + (x_By_C - x_Cy_B)\right|$$

$$= \frac{1}{2}\left|x_Ay_B - x_By_A + x_By_C - x_Cy_B + x_Cy_A - x_Ay_C\right|$$

Biện luận bằng cách so sánh tổng diện tích: ΔMAB + ΔMBC + ΔMCA với diện tích ΔABC.

```
A(xA, yA)? 0 5 ↵
B(xB, yB)? 3 0 ↵
C(xC, yC)? 4 7 ↵
M(xM, yM)? 2 6 ↵
M nam tren canh tam giac ABC
```

Bài giải: xem trang 68

Bài 6: Viết chương trình nhập vào ba số nguyên. Hãy in ba số này ra màn hình theo thứ tự tăng dần và chỉ dùng tối đa một biến phụ.

```
Nhap a, b, c: 5 3 4 ↵
Tang dan: 3 4 5
```

Bài giải: xem trang 69

Bài 7: Viết chương trình giải phương trình bậc 1: `ax + b = 0` (`a`, `b` nhập từ bàn phím). Xét tất cả các trường hợp có thể.

```
Nhap a, b: 4 -3
x = 0.75
```

Bài giải: xem trang 70

Bài 8: Viết chương trình giải phương trình bậc 2: $ax^2 + bx + c = 0$ (`a`, `b`, `c` nhập từ bàn phím). Xét tất cả các trường hợp có thể.

Nghiệm của phương trình bậc 2: $ax^2 + bx + c = 0$ $(a \neq 0)$

$x = \dfrac{-b \pm \sqrt{\Delta}}{2a}$, với delta: $\Delta = b^2 - 4ac$

```
Nhap a, b, c: 2 1 -4 ↵
x1 = -6.74456
x2 = 4.74456
```

Bài giải: xem trang 71

Bài 9: Viết chương trình nhập vào số `x` chỉ số đo của một góc, tính bằng phút. Cho biết nó thuộc góc vuông thứ bao nhiêu của vòng tròn lượng giác.
Tính `cos(x)`, dùng hàm do `math.h` cung cấp.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

60’ = 1°. Công thức chuyển đổi giữa độ và radian: 1 radian = $\frac{180}{\pi}$ degree

```
Nhap so do x cua goc (phut): 12345 ↵
x thuoc goc vuong thu 3
cos(x) = -0.900698
```

Bài giải: xem trang 72

Bài 10: Số bảo hiểm xã hội của Canada (SIN - Canadian Social Insurance Number) là một số có 9 chữ số, được kiểm tra tính hợp lệ như sau:
- Số phải nhất (vị trí 1, tính từ phải sang), là số kiểm tra (check digit).
- Trọng số được tính từ phải qua trái (không tính check digit), bằng `s1 + s2`:
+ `s1` là tổng các số có vị trí lẻ.
+ Các số có vị trí chẵn nhân đôi. Nếu kết quả nhân đôi có hai chữ số thì kết quả là tổng của hai chữ số này. `s2` là tổng các kết quả.
SIN hợp lệ có tổng trọng số với số kiểm tra chia hết cho `10`.
Ví dụ: SIN `193456787`
- Số kiểm tra là `7` (số tô đậm).
- Trọng số là tổng của `s1` và `s2`, với:
+ `s1 = 1 + 3 + 5 + 7 = 16`
+ Các số có vị trí chẵn nhân đôi:
`(9 * 2) (4 * 2) (6 * 2) (8 * 2)` ⇒ `18 8 12 16`
`s2 = (1 + 8) + 8 + (1 + 2) + (1 + 6) = 27`
Trọng số bằng `s1 + s2 = 16 + 27 = 43`.
Vì tổng trọng số với số kiểm tra `43 + 7 = 50` chia hết cho `10` nên số SIN hợp lệ.
Viết chương trình nhập một số SIN. Kiểm tra xem số SIN đó có hợp lệ hay không. Nhập 0 để thoát.

```
SIN (0 de thoat): 193456787 ↵
SIN hop le!
SIN (0 de thoat): 193456788 ↵
SIN khong hop le!
SIN (0 de thoat): 0 ↵
```

Bài giải: xem trang 72

Bài 11: Viết trò chơi bao - đá - kéo với luật chơi: bao thắng đá, đá thắng kéo, kéo thắng bao. Người dùng nhập vào một trong ba ký tự `b` (bao), `d` (đá), `k` (kéo); máy tính sinh ngẫu nhiên một trong ba ký tự trên, thông báo kết quả chơi.

```
Nhap ky tu (b-d-k), ky tu khac de thoat: b ↵
Computer: d
Ty so human - computer: 1 - 0
Nhap ky tu (b-d-k), ky tu khac de thoat: k ↵
Computer: d
Ty so human - computer: 1 - 1
Nhap ky tu (b-d-k), ky tu khac de thoat: 0 ↵
```

Bài giải: xem trang 73

Bài 12: Viết chương trình giải hệ phương trình 2 ẩn:

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

$$\begin{cases} a_1x + b_1y = c_1 \\ a_2x + b_2y = c_2 \end{cases}$$

Các hệ số a_1, a_2, b_1, b_2, c_1, c_2 nhập từ bàn phím. Xét tất cả các trường hợp cụ thể.

Công thức Cramer[2] dùng tính hệ phương trình 2 ẩn:

$$D = \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} \quad D_x = \begin{vmatrix} c_1 & b_1 \\ c_2 & b_2 \end{vmatrix} \quad D_y = \begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}$$

Nếu $D \neq 0, x = \frac{D_x}{D}, y = \frac{D_y}{D}$

```
Nhap a1, b1, c1: 1 2 3 ↵
Nhap a2, b2, c2: 4 5 6 ↵
x = -1
y = 2
```

Bài giải: xem trang 74

Bài 13: Viết chương trình nhập vào ngày, tháng, năm. Kiểm tra ngày và tháng nhập có hợp lệ hay không. Tính thứ trong tuần của ngày đó.

Năm nhuận (leap year) tính theo lịch Gregorian (từ 1582): năm phải chia hết cho 4 và không chia kết cho 100, hoặc năm phải chia hết cho 400.
Thứ trong tuần tính theo công thức Zeller[3]:

```
dayofweek = (d + y + y/4 - y/100 + y/400 + (31 * m)/12) % 7
```

với:
```
a = (14 - month)/12
y = year - a
m = month + 12*a - 2
```

`dayofweek`: `0` (chúa nhật), `1` (thứ hai), `2` (thứ ba), …

```
Nhap ngay, thang va nam: 20 4 1976 ↵
Hop le
Thu 3
```

Bài giải: xem trang 75

Bài 14: Viết chương trình nhập vào ngày, tháng, năm (giả sử nhập đúng, không cần kiểm tra hợp lệ). Tìm ngày, tháng, năm của ngày tiếp theo.
Tương tự, tìm ngày, tháng, năm của ngày trước đó.

```
Nhap ngay, thang, nam: 28 2 2000 ↵
Ngay mai: 29/02/2000
Nhap ngay, thang, nam: 1 1 2001 ↵
Hom qua: 31/12/2000
```

Bài giải: xem trang 76

Bài 15: Viết chương trình nhập vào ngày, tháng, năm (giả sử nhập đúng, không cần kiểm tra hợp lệ). Tìm xem ngày đó là ngày thứ bao nhiêu trong năm.

Nếu không dùng vòng lặp, có thể dùng công thức sau:

```
sum = (int) (30.42 * (month - 1)) + day
```

[2] Gabriel Cramer (1704 - 1752)
[3] Julius Christian Johannes Zeller (1824 - 1899)

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
Nếu month = 2, hoặc năm nhuận và month > 2 thì sum = sum + 1
Nếu 2 < month < 8 thì sum = sum - 1
```

```
Nhap ngay, thang, nam: 4 4 2000 ↵
Ngay thu: 95
```

Bài giải: xem trang 77

Bài 16: Viết chương trình nhập vào một năm (> 1582), in lịch của năm đó. Tính thứ cho ngày đầu năm bằng công thức Zeller (bài 14, trang 6).

```
Nhap nam: 2008 ↵
Thang 1
  S  M  T  W  T  F  S
        1  2  3  4  5
  6  7  8  9 10 11 12
 13 14 15 16 17 18 19
 20 21 22 23 24 25 26
 27 28 29 30 31
...
Thang 12
  S  M  T  W  T  F  S
     1  2  3  4  5  6
  7  8  9 10 11 12 13
 14 15 16 17 18 19 20
 21 22 23 24 25 26 27
 28 29 30 31
```

Bài giải: xem trang 78

Bài 17: Viết chương trình tạo lịch trực cho 5 bạn: `A`, `B`, `C`, `D`, `E`. Nhập năm và thứ (`0` - `6`, `0` là Chúa Nhật, `1` là thứ Hai, …) cho ngày đầu năm. Sau đó nhập một tháng trong năm và in lịch trực của tháng đó. Lưu ý 5 bạn trực lần lượt theo thứ tự trên, ngày Chúa nhật không trực và bạn `A` sẽ trực ngày đầu tiên của năm.

```
Nhap nam: 2006 ↵
Nhap thu cho ngay dau tien cua nam: 0 ↵
Nhap thang: 5 ↵
    Sun     Mon     Tue     Wen     Thu     Fri     Sat
         1 [C]   2 [D]   3 [E]   4 [A]   5 [B]   6 [C]
  7 [ ]   8 [D]   9 [E] 10 [A] 11 [B] 12 [C] 13 [D]
 14 [ ] 15 [E] 16 [A] 17 [B] 18 [C] 19 [D] 20 [E]
 21 [ ] 22 [A] 23 [B] 24 [C] 25 [D] 26 [E] 27 [A]
 28 [ ] 29 [B] 30 [C] 31 [D]
```

Bài giải: xem trang 80

Bài 18: Viết chương trình nhập vào số giờ, xuất ra số tương đương tính theo tuần, theo ngày và theo giờ.

```
Nhap so gio: 1000 ↵
5 tuan, 6 ngay, 16 gio
```

Bài giải: xem trang 81

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 19: Nhập vào thời điểm 1 và thời điểm 2. Tìm thời gian trải qua giữa hai thời điểm này tính bằng giờ, phút, giây.

```
Nhap gio, phut, giay [1]: 3 28 47 ↵
Nhap gio, phut, giay [2]: 5 40 12 ↵
Hieu thoi gian: 2 gio 11 phut, 25 giay
```

Bài giải: xem trang 81

Bài 20: Viết chương trình nhập số kW điện đã tiêu thụ. Tính tiền điện phải trả, biết rằng khung giá điện như sau:

0kW	100kW	250kW	350kW
500đ/kW	800đ/kW	1000đ/kW	1500đ/kW

```
Nhap so kW tieu thu: 4321 ↵
Chi phi: 6226500
```

Bài giải: xem trang 82

Bài 21: Trong kỳ thi tuyển, một thí sinh sẽ trúng truyển nếu có điểm tổng kết lớn hơn hoặc bằng điểm chuẩn và không có môn nào điểm 0.
- Điểm tổng kết là tổng điểm của 3 môn thi và điểm ưu tiên.
- Điểm ưu tiên bao gồm điểm ưu tiên theo khu vực và điểm ưu tiên theo đối tượng.

Khu vực			Đối tượng		
A	B	C	1	2	3
2	1	0.5	2.5	1.5	1

Viết chương trình nhập: điểm chuẩn của hội đồng, điểm 3 môn thi của thí sinh, khu vực (nhập `X` nếu không thuộc khu vực ưu tiên) và đối tượng dự thi (nhập `0` nếu không thuộc đối tượng ưu tiên). Cho biết thí sinh đó đậu hay rớt và tổng số điểm đạt được.

```
Nhap diem chuan: 15.5 ↵
Nhap diem 3 mon thi: 4.5 3.4 3.6 ↵
Nhap khu vuc (A, B, C, X): B ↵
Nhap doi tuong (1, 2, 3, 0): 1 ↵
Rot [15]
```

Bài giải: xem trang 83

Bài 22: Viết chương trình liệt kê, đếm và tính tổng các ước số của số nguyên dương `n` (`n` nhập từ bàn phím).

```
Nhap n: 1966 ↵
Cac uoc so: 1 2 983 1966
Co 4 uoc so, tong la: 2952
```

Bài giải: xem trang 83

Bài 23: Viết chương trình tìm các số hoàn hảo (perfect number) nhỏ hơn một số nguyên dương `n` cho trước. Biết số hoàn hảo là số nguyên dương, bằng tổng các *ước số thực sự* của nó (ví dụ: 28 = 14 + 7 + 4 + 2 + 1).

```
Nhap n: 10000 ↵
Cac so hoan hao nho hon 10000: 6 28 496 8128
```

Bài giải: xem trang 84

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 24: Nhập vào một số tự nhiên `n` (`n` khai báo kiểu `unsigned long`)

a. Số tự nhiên `n` có bao nhiêu chữ số.
b. Hãy tìm chữ số cuối cùng của `n`.
c. Hãy tìm chữ số đầu tiên của `n`.
d. Tính tổng các chữ số của `n`.
e. Hãy tìm số đảo ngược của `n`.

```
Nhap n: 43210 ↵
43210 co 5 chu so
Chu so cuoi cung la: 0
Chu so dau tien la: 4
Tong cac chu so la: 10
So dao nguoc la: 1234
```

Bài giải: xem trang 84

Bài 25: Nhập vào hai số nguyên dương `a`, `b`. Tính ước số chung lớn nhất và bội số chung nhỏ nhất của `a`, `b`.

USCLN: (Greatest Common Divisor) gcd(a, b) = max{k | k\a ∧ k\b}
BSCNN: (Least Common Multiple) lcd(a, b) = min{k | k>0, a\k ∧ b\k}

USCLN(a, b): + Cho gcd bằng a hoặc b
+ Trừ dần gcd cho đến khi cả a và b đều chia hết cho gcd
+ USCLN (a, b) = gcd
BSCNN(a, b): + Cho lcm bằng a hoặc b
+ Tăng dần lcm cho đến khi lcm chia hết cho cả a và b
+ BSCNN (a, b) = lcm

```
Nhap cap (a, b): 12 8 ↵
USCLN (a, b): 4
BSCNN (a, b): 24
```

Bài giải: xem trang 86

Bài 26: Nhập vào tử số, mẫu số (đều khác 0) của một phân số. Hãy rút gọn phân số này. Chọn dạng xuất thích hợp trong trường hợp mẫu số bằng 1 và phân số có dấu.

Để rút gọn một phân số, chia cả tử số và mẫu số cho USLCN của tử số và mẫu số.

```
Nhap tu so, mau so: -3 -15 ↵
Rut gon: 1/5
Nhap tu so, mau so: 8 -2 ↵
Rut gon: -4
```

Bài giải: xem trang 88

Bài 27: Nhập vào một số nguyên dương `n`, phân tích `n` thành các thừa số nguyên tố.

```
Nhap n: 12345 ↵
3 * 5 * 823
```

Bài giải: xem trang 89

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 28: Viết chương trình mô phỏng hàm `ROUND` của Microsoft Excel, dùng làm tròn một số `double` với một số `n` cho trước.

- Nếu `n > 0`, số làm tròn sẽ có `n` chữ số phần thập phân.
- Nếu `n = 0`, số làm tròn sẽ là số nguyên gần nhất.
- Nếu `n < 0`, số làm tròn là số nguyên làm tròn từ vị trí thứ `n` tính từ phải sang.

```
Nhap so thuc x: 3.1415926535 ↵
Do chinh xac: 7 ↵
3.1415927
Nhap so thuc x: -4.932 ↵
Do chinh xac: 0 ↵
-5
Nhap so thuc x: 21.5 ↵
Do chinh xac: -1 ↵
20
```

Bài giải: xem trang 90

Bài 29: Lập bảng so sánh hai thang đo nhiệt độ Fahrenheit và Celsius[4] trong:
- Đoạn `[0°C, 10°C]`, bước tăng `1°C`.
- Đoạn `[32°F, 42°F]`, bước tăng `1°F`.

Công thức chuyển đổi Fahrenheit - Celcius:
`5(F - 32) = 9C`

```
Celcius  Fahrenheit      Fahrenheit  Celcius
      0       32.00              32     0.00
      1       33.80              33     0.56
      2       35.60              34     1.11
      3       37.40              35     1.67
      4       39.20              36     2.22
      5       41.00              37     2.78
      6       42.80              38     3.33
      7       44.60              39     3.89
      8       46.40              40     4.44
      9       48.20              41     5.00
     10       50.00              42     5.56
```

Bài giải: xem trang 91

Bài 30: Viết chương trình nhập lãi xuất năm `r` (%), tiền vốn `p` và thời hạn gởi tiền `n` (năm). Mỗi trị nhập phải cách nhau bởi dấu “,”. In ra vốn tích lũy `a` của từng năm. Chương trình có kiểm tra nhập thiếu hoặc nhập lỗi.

$a = p(1 + r)^n$

Trong đó, `a` (mount) là vốn tích lũy được, `p` (principal) là vốn gốc, `r` là (rate) lãi suất và `n` là số năm đầu tư.

[4] Gabriel Fahrenheit (1686 - 1736) và Anders Celsius (1701 - 1744)

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
Nhap lai suat, tien von, thoi han: 0.027, 15000, 3 ↵
Lai suat: 2.7%
Von ban dau: 15000
Thoi han: 3 nam
Nam        Von
  1      15405
  2    15820.9
  3    16248.1
```

Bài giải: xem trang 91

Bài 31: Viết chương trình in bảng cửu chương từ 2 đến 9 ra màn hình.

```
Bang cuu chuong
|2x 1= 2|3x 1= 3|4x 1= 4|5x 1= 5|6x 1= 6|7x 1= 7|8x 1= 8|9x 1= 9|
|2x 2= 4|3x 2= 6|4x 2= 8|5x 2=10|6x 2=12|7x 2=14|8x 2=16|9x 2=18|
|2x 3= 6|3x 3= 9|4x 3=12|5x 3=15|6x 3=18|7x 3=21|8x 3=24|9x 3=27|
|2x 4= 8|3x 4=12|4x 4=16|5x 4=20|6x 4=24|7x 4=28|8x 4=32|9x 4=36|
|2x 5=10|3x 5=15|4x 5=20|5x 5=25|6x 5=30|7x 5=35|8x 5=40|9x 5=45|
|2x 6=12|3x 6=18|4x 6=24|5x 6=30|6x 6=36|7x 6=42|8x 6=48|9x 6=54|
|2x 7=14|3x 7=21|4x 7=28|5x 7=35|6x 7=42|7x 7=49|8x 7=56|9x 7=63|
|2x 8=16|3x 8=24|4x 8=32|5x 8=40|6x 8=48|7x 8=56|8x 8=64|9x 8=72|
|2x 9=18|3x 9=27|4x 9=36|5x 9=45|6x 9=54|7x 9=63|8x 9=72|9x 9=81|
|2x10=20|3x10=30|4x10=40|5x10=50|6x10=60|7x10=70|8x10=80|9x10=90|
```

Bài giải: xem trang 92

Bài 32: Cho n_i là một số nguyên dương, với định nghĩa:

$$n_{i+1} = \begin{cases} n_i / 2 & n_i = 2k + 1 \\ 3n_i + 1 & n_i = 2k \end{cases}$$

Chuỗi trên sẽ ngừng khi n_i có trị 1. Các số được sinh ra gọi là hailstones (mưa đá) và quá trình trên đã được chứng minh là luôn luôn dừng. Viết chương trình sinh ra chuỗi hailstones với n_i nhập vào từ bàn phím.

```
Nhap n: 15 ↵
   15   46   23   70   35  106
   53  160   80   40   20   10
    5   16    8    4    2    1
Hailstones sinh duoc: 18
Tiep (y/n)? n ↵
```

Bài giải: xem trang 93

Bài 33: Số tự nhiên có n chữ số là một số Armstrong (còn gọi là narcissistic numbers hoặc pluperfect digital invariants - PPDI) nếu tổng các lũy thừa bậc n của các chữ số của nó bằng chính nó. Hãy tìm tất cả các số Armstrong có 3, 4 chữ số.
Ví dụ: 153 là số Armstrong có 3 chữ số vì: $1^3 + 5^3 + 3^3 = 153$

```
So Armstrong co 3, 4 chu so:
153 370 371 407 1634 8208 9474
```

Bài giải: xem trang 93

Bài 34: Dùng công thức hình thang, tính gần đúng tích phân xác định sau với độ chính xác 10^{-6}:

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

$$\int_0^{\pi/2} \sin^2(x)\cos(x)dx$$

Kiểm chứng với cách tính trực tiếp:

$$\int_0^{\pi/2} \sin^2(x)\cos(x)dx = \frac{\sin^3(x)}{3}\Big|_0^{\pi/2} = \frac{1}{3}\left(\sin^3\left(\frac{\pi}{2}\right) - \sin^3(0)\right) = \frac{1}{3}\sin^3\left(\frac{\pi}{2}\right)$$

Để tính gần đúng tích phân xác định, người ta thường dùng công thức hình thang (trapezoidal rule) như sau:

$$\int_a^b f(x)dx \approx h\left[\frac{f(x_0)+f(x_1)}{2} + \frac{f(x_1)+f(x_2)}{2} + \ldots + \frac{f(x_{n-1})+f(x_n)}{2}\right]$$

$$= h\left[\frac{f(x_0)+f(x_n)}{2} + f(x_1) + \ldots + f(x_{n-1})\right] = I_n$$

với: $h = \frac{b-a}{n}$, $x_i = a + ih$

Để đạt độ chính xác, chọn n_0 tùy ý, sau đó tính I_n với $n = n_0, 2n_0, 4n_0 \ldots$ Việc tính toán dừng lại khi $|I_{2n} - I_n| / 3 < e$ (e là độ chính xác).

```
Ket qua  : 0.333333
Doi chung: 0.333333
```

Bài giải: xem trang 94

Bài 35: Viết chương trình kiểm tra một số nguyên dương n có là số nguyên tố hay không. Nếu không thì phải xác định số nguyên tố gần n nhất và bé hơn n.

Số nguyên tố n là một số nguyên lớn hơn 1, chỉ có hai ước số (chỉ chia hết): 1 và chính nó.

Để xác định n là số nguyên tố, chỉ cần kiểm tra n không có ước số từ 2 đến $\sqrt{n}$; do mọi hợp số (số nguyên lớn hơn 1 không phải là số nguyên tố) n đều có ước số nguyên tố nhỏ hơn $\sqrt{n}$ [5].

```
Nhap n: 822 ↵
822 khong la so nguyen to
So nguyen to be hon gan nhat: 821
```

Bài giải: xem trang 95

Bài 36: Viết chương trình in ra n số nguyên tố đầu tiên (n nhập từ bàn phím).

```
Nhap n: 15 ↵
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
```

Bài giải: xem trang 97

[5] Vì n là hợp số, ta có thể viết n = a*b, trong đó a, b là các số nguyên với 1 < a ≤ b < n. Rõ ràng phải có a hoặc b không vượt quá $\sqrt{n}$, giả sử là b. Ước số nguyên tố của b cũng là ước số nguyên tố của n.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 37: Viết chương trình nhập số nguyên dương `n`. Tìm số nguyên dương `m` lớn nhất sao cho: `1 + 2 + … + m < n`.

```
Nhap n: 22 ↵
1 + 2 + 3 + 4 + 5 + 6 = 21 < 22
m = 6
```

Bài giải: xem trang 98

Bài 38: Nhập vào một số tiền `n` (nghìn đồng, `n > 5`) nguyên dương. Đổi số tiền này ra ba loại tiền giấy `1000VNĐ`, `2000VNĐ`, `5000VNĐ`.
Tìm phương án đổi tiền sao cho loại tiền `2000VNĐ` chiếm hơn phân nửa số tờ bạc phải đổi ít nhất.

```
Nhap n (nghin dong, n > 5): 137 ↵
( 0, 21, 19 ): 40
```

Bài giải: xem trang 99

Bài 39: Tìm các bộ số nguyên dương (`x, y, z`) (`x, y, z < 100`) là 3 số nguyên liên tiếp hoặc 3 số chẵn liên tiếp, thỏa mãn công thức Pithagoras[6]: $x^2 + y^2 = z^2$

```
(3, 4, 5): ba so nguyen lien tiep
(6, 8, 10): ba so chan lien tiep
```

Bài giải: xem trang 99

Bài 40: Tìm các bộ (trâu đứng, trâu nằm, trâu già) thỏa mãn bài toán cổ:

Trăm trâu ăn trăm bó cỏ
Trâu đứng ăn năm
Trâu nằm ăn ba
Lụ khụ trâu già
Ba con một bó

Thử tìm cách giảm số vòng lặp khi tính toán xuống.

```
(4, 18, 78)
(8, 11, 81)
(12, 4, 84)
```

Bài giải: xem trang 100

Bài 41: Viết chương trình tìm cách thay thế các dấu hỏi (`?`) bởi các dấu 4 phép tính số học `+`, `-`, `*`, `/`, trong biểu thức dưới đây sao cho biểu thức có giá trị bằng 36.

```
((((1 ? 2) ? 3) ? 4) ? 5) ? 6
```

```
((((1 - 2) + 3) + 4) * 5) + 6 = 36
((((1 - 2) * 3) + 4) + 5) * 6 = 36
((((1 * 2) + 3) - 4) + 5) * 6 = 36
((((1 / 2) * 3) * 4) * 5) + 6 = 36
```

Bài giải: xem trang 100

[6] Pythagoras (582 BC - 507 BC)

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 42: Giả thuyết Goldbach[7] (Goldbach's conjecture) cho rằng: một số nguyên tố n bất kỳ (n > 5) đều có thể khai triển thành tổng của ba số nguyên tố khác.
Viết chương trình kiểm chứng giả thuyết Goldbach với n < 1000.

```
Co 165 so nguyen to n (5 < n < 1000)
  7 = 2 + 2 + 3
 11 = 2 + 2 + 7
      ...
997 = 3 + 3 + 991
Kiem chung dung voi 165 so nguyen to
```

Bài giải: xem trang 101

Bài 43: Tìm số Fibonacci[8] thứ n (n < 40), dùng vòng lặp (không dùng đệ quy).

Số Fibonacci thứ n: $F(n) = \begin{cases} 1 & n = 1, 2 \\ F(n-2) = F(n-1) & n > 2 \end{cases}$

```
Nhap n (n < 40): 24 ↵
Fi(24) = 46368
```

Bài giải: xem trang 102

Bài 44: Dùng vòng lặp lồng, viết chương trình in ra tam giác cân đặc và rỗng, tạo từ các dấu sao (*), có độ cao là n nhập từ bàn phím.

```
Nhap n: 4 ↵
      *
    * * *
  * * * * *
* * * * * * *
      *
    *   *
  *       *
* * * * * * *
```

Bài giải: xem trang 103

Bài 45: Dùng vòng lặp lồng, với n (n < 5) nhập từ bàn phím, viết chương trình in hai tam giác đối đỉnh bằng số, tăng theo cột từ 1 đến 2n - 1.

```
Nhap n (n < 5): 3 ↵
 1       5
 1 2   4 5
 1 2 3 4 5
 1 2   4 5
 1       5
```

Bài giải: xem trang 104

Bài 46: Viết chương trình kiểm tra hai vế của công thức sau, với n cho trước:

$$\sum_{i=3}^{n} i^3 = \frac{n^2(n+1)^2}{4}$$

[7] Christian Goldbach (1690 - 1764)
[8] Leonardo Fibonacci (1170 - 1250)

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
Nhap n: 50 ↵
Ve trai = 1625625
Ve phai = 1625625
```

Bài giải: xem trang 104

Bài 47: Với n cho trước, tính tổng S, biết:

Nếu n chẵn: S = 2 + 4 + 6 + … + n

Nếu n lẻ: S = 1 + 2 + 3 + … + n

```
Nhap n: 120 ↵
S = 3660
```

Bài giải: xem trang 105

Bài 48: Với số nguyên n cho trước, tìm ước số lẻ lớn nhất của n và ước số lớn nhất của n là lũy thừa của 2.

```
Nhap n: 384 ↵
US le lon nhat: 3
US lon nhat la luy thua cua 2: 128
```

Bài giải: xem trang 105

Bài 49: Viết chương trình tính căn số liên tục sau:

$$S = \sqrt[n+1]{n + \sqrt[n]{n - 1 + \sqrt[n-1]{n - 2 + \ldots + \sqrt[3]{2 + \sqrt{1}}}}}$$

```
Nhap n: 10 ↵
Ket qua: 1.24624
```

Bài giải: xem trang 107

Bài 50: Phân số liên tục (continued fraction) ký hiệu $[b_1, b_2, \ldots, b_k]$, có dạng:

$$\frac{s}{t} = \cfrac{1}{b_1 + \cfrac{1}{b_2 + \cfrac{1}{\cdots \; b_{k-1} + \cfrac{1}{b_k}}}}$$

$b_1, b_2, \ldots, b_k$ là các số tự nhiên. Cho s và t, viết chương trình tìm $[b_1, b_2, \ldots, b_k]$.

Mỗi phân số hữu tỷ $\frac{s}{t}$ ($0 < s < t$ là các số tự nhiên) đều có thể đưa về dạng phân số liên tục bằng thuật toán sau:

1. Chia t cho s, được a dư r: $t = a * s + r$. Suy ra: $\frac{s}{t} = \frac{1}{\frac{t}{s}} = \frac{1}{a + \frac{r}{s}}$

2. Đặt $b_1 = a$, rồi tiếp tục biến đổi $\frac{r}{s}$ cho đến khi số dư r bằng 0.

```
Nhap s, t (0 < s < t): 123 1234 ↵
[10, 30, 1, 3]
```

Bài giải: xem trang 107

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 51: Viết chương trình tính phân số liên tục sau:

$$F = x + \cfrac{1}{x + \cfrac{2}{x + \cfrac{4}{\cdots \; x + \cfrac{128}{x + \cfrac{256}{x}}}}} \quad \text{(x là số thực khác 0)}$$

```
Nhap x: 2.4 ↵
F = 2.73649
```

Bài giải: xem trang 108

Bài 52: Cho số tự nhiên n, hãy tính F_n biết: $F_n = \sum_{i=1}^{n} \frac{1}{n^2 + i}$

```
Nhap n: 12 ↵
Fn = 0.0797762
```

Bài giải: xem trang 109

Bài 53: Viết chương trình tính sin(x) với độ chính xác 10^{-4} theo chuỗi Taylor[9] (Taylor series):

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

```
Nhap x (radian): 2.7 ↵
cong thuc Taylor: sin(2.70) = 0.4274
sin() cua math.h: sin(2.70) = 0.4274
```

Bài giải: xem trang 109

Bài 54: Dùng vòng lặp, tính tổ hợp chập k của n (k < n < 25):

$$C_n^k = \frac{n!}{k!\,(n-k)!}$$

Kiểm chứng công thức $C_n^k = C_n^{n-k}$

```
Nhap n, k (k < n < 25): 20 5 ↵
C( k, n ): 15504
C( n-k, n ): 15504
```

Bài giải: xem trang 110

Bài 55: Tính căn bậc hai của một số nguyên dương x bằng thuật toán Babylonian. Kiểm tra kết quả với hàm chuẩn sqrt().

Thuật toán Babylonian dùng tính căn bậc hai của một số nguyên dương x:
1. Đặt y = 1.0
2. Thay y với trung bình cộng của y và x/y
3. Lặp lại bước 2 đến khi y không còn thay đổi (y xấp xỉ bằng x/y)
4. Trả về y

[9] Brook Taylor (1685 - 1731)

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
Nhap x (x > 0): 7 ↵
thuat toan babylonian: 2.64575
ham sqrt() cua math.h: 2.64575
```

Bài giải: xem trang 111

Bài 56: Viết chương trình nhập vào một số nguyên `n` có dấu, in ra dạng hiển thị nhị phân và thập lục phân của `n`.

Để xác định một bit tại vị trí bất kỳ, dùng mặt nạ (mask) AND, kết hợp với toán tử AND bitwise (&):
Mặt nạ thường là một dãy bit 0, với bit 1 được bật tại vị trí cần kiểm tra.

```
        10 1 1010                    10 0 1010
mask    00 1 0000            mask    00 1 0000
        00 1 0000                    00 0 0000
     (≠0) ⇒ bit 1                 (=0) ⇒ bit 0
```

Khi cần xác định bit tại vị trí khác, dùng toán tử dịch bit để di chuyển bit 1 của mặt nạ; hoặc dịch chuyển số kiểm tra để bit cần kiểm tra đến đúng vị trí bit 1 của mặt nạ.
Không dùng cách chia 2 để chuyển số nguyên thành số nhị phân rồi lấy từng bit vì số nguyên âm lưu ở dạng số bù 2 (two's complement).

```
Nhap n: -5678 ↵
-5678 = 11111111 11111111 11101001 11010010
Hex: FFFFE9D2
```

Bài giải: xem trang 111

Bài 57: Bit parity là bit thêm vào một nhóm mã được truyền đi, dùng để phát hiện lỗi một bit đơn trong quá trình truyền. Bit parity chẵn (even parity) là bit có trị được chọn sao cho tổng số bit `1` trong một nhóm mã kể cả bit parity là một số chẵn. Viết chương trình nhập vào một số nguyên `n`. Xác định bit parity chẵn của `n`.

Bit parity chẵn của `n` sẽ bằng `0` nếu số các bit `1` là số chẵn và bằng `1` nếu số các bit `1` là số lẻ.

```
Nhap n: 13579 ↵
Even parity bit = 1
```

Bài giải: xem trang 112

MẢNG

Bài 58: Viết chương trình thực hiện thuật toán sàng Erastosthenes[10] (Sieve of Erastosthenes) để in ra các số nguyên tố nhỏ hơn số `n` cho trước (`n < 100`).

Sàng Erastosthenes: viết các số nguyên từ 2 đến n. Khoanh tròn 2; gạch chéo tất cả những bội số khác của 2. Lặp lại bằng cách khoanh tròn số nhỏ nhất chưa được khoanh tròn, không bị gạch chéo; gạch chéo tất cả những

[10] Eratosthenes (276 BC - 194 BC)

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

bội số của nó. Khi không còn số nào để khoanh tròn hoặc gạch chéo thì dừng. Tất cả những số được khoanh tròn là số nguyên tố.

```
Nhap n: 64 ↵
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61
```

Bài giải: xem trang 113

Bài 59: Nhập vào năm Dương lịch, xuất tên năm Âm lịch. Xuất năm Dương lịch kế tiếp có cùng tên năm Âm lịch. Biết bánh xe tính hai chu kỳ Can - Chi như sau:

Năm có cùng tên Âm lịch với năm y là `y ± k * 60` (`60` là `BSCNN` của hai chu kỳ `10` và `12`). Mốc tính Can Chi, lấy năm `0` là năm Canh Thân.

```
Nhap nam: 2000 ↵
2000 - Canh Thin
2060 - Canh Thin
```

Bài giải: xem trang 114

Bài 60: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ngẫu nhiên mảng một chiều `n` phần tử nguyên (`n` chẵn) có giá trị chứa trong đoạn `[-100, 100]` và xuất mảng.
b. Viết hàm thực hiện việc trộn hoàn hảo (perfect shuffle) một mảng: sao cho các phần tử của một nửa mảng sau xen kẽ với các phần tử của một nửa mảng đầu. Xuất mảng sau khi trộn.
c. Xác định số lần trộn hoàn hảo để mảng trở về như ban đầu.

```
Nhap n (n chan): 12
-33 62 -12 34 -89 65 -3 -96 86 89 39 35
-33 -3 62 -96 -12 86 34 89 -89 39 65 35
Can 10 lan shuffle de mang tro ve ban dau
```

Bài giải: xem trang 114

Bài 61: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ngẫu nhiên mảng một chiều `n` phần tử nguyên có giá trị chứa trong đoạn `[-100, 100]` và xuất mảng.
b. Tính tổng các số nguyên dương có trong mảng.
c. Xóa phần tử có chỉ số `p` (`p` nhập từ bàn phím) trong mảng.

```
Nhap n [1, 99]: 10 ↵
69 -41 48 22 -34 100 -14 70 66 -29
Tong cac so nguyen duong = 375
Nhap p [0, 9]: 4 ↵
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
69 -41 48 22 100 -14 70 66 -29
```

Bài giải: xem trang 116

Bài 62: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ngẫu nhiên mảng một chiều `n` phần tử nguyên dương có giá trị chứa trong đoạn `[10, 20]` và xuất mảng.
b. Kiểm tra xem tổng các số chẵn ở vị trí lẻ có bằng tổng các số lẻ ở vị trí chẵn hay không?
c. Xác định xem mảng có cặp số nguyên tố cùng nhau (coprime) nào không.

Hai số nguyên dương `a` và `b` được gọi là hai số nguyên tố cùng nhau nếu ước số chung lớn nhất của hai số `a` và `b` là `1`.

```
Nhap n [1, 99]: 5 ↵
14 14 11 16 12
Tong le vi tri chan (30) khac tong chan vi tri le (11)
Cac cap nguyen to cung nhau:
(14, 11)
(11, 16)
(11, 12)
```

Bài giải: xem trang 117

Bài 63: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ngẫu nhiên mảng một chiều `n` phần tử nguyên có giá trị chứa trong đoạn `[-100, 100]` và xuất mảng.
b. Đếm số phần tử chia hết cho 4 và có chữ số tận cùng là 6.
c. Thay các phần tử lẻ bằng 2 lần giá trị của nó.

```
Nhap n [1, 99]: 10 ↵
70 -67 22 -87 34 16 -34 -58 76 -78
Co 2 phan tu chia het cho 4, tan cung 6
Nhan doi phan tu le:
70 -134 22 -174 34 16 -34 -58 76 -78
```

Bài giải: xem trang 118

Bài 64: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo mảng một chiều `n` phần tử nguyên có giá trị nhập vào từ bàn phím.
b. Hãy đếm số các phần tử có trị là lũy thừa của 2 có trong mảng.
c. Nhập `x` nguyên, xóa các phần tử trong mảng có trị trùng với `x`.

Một số là lũy thừa của 2 nếu số đó có bit 1 duy nhất là bit MSB (Most Significant Bit). Ví dụ: 2^8 = 256 = **1** 0000 0000

```
Nhap n [1, 99]: 10 ↵
Nhap 10 phan tu:
2 -5 4 7 9 -8 32 16 11 4 ↵
Co 5 so la luy thua cua 2
Nhap x: 4 ↵
2 -5 7 9 -8 32 16 11
```

Bài giải: xem trang 119

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 65: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo mảng một chiều n phần tử nguyên có giá trị nhập vào từ bàn phím.
b. Tính trung bình cộng của các số nguyên âm lẻ có trong mảng.
c. Xóa các phần tử có trị trùng nhau trong mảng, chỉ chừa lại một phần tử.

```
Nhap n [1, 99]: 10 ↵
Nhap 10 phan tu:
2 2 -3 7 4 -5 4 9 -1 -1 ↵
Trung binh cong nguyen am le = -2.50
2 -3 7 4 -5 9 -1
```

Bài giải: xem trang 120

Bài 66: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ngẫu nhiên mảng một chiều n phần tử nguyên có giá trị chứa trong đoạn `[-100, 100]` và xuất mảng.
b. Dùng một vòng lặp, tìm phần tử có trị nhỏ nhất và lớn nhất của mảng.
c. Xóa các phần tử trong mảng có trị trùng với giá trị lớn nhất của mảng, trừ phần tử tìm được đầu tiên.

```
Nhap n [1, 99]: 10 ↵
21 1 -68 24 22 -76 -69 0 24 -84
max = 24
min = -84
21 1 -68 24 22 -76 -69 0 -84
```

Bài giải: xem trang 121

Bài 67: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ngẫu nhiên mảng một chiều n phần tử nguyên có giá trị chứa trong đoạn `[-100, 100]` và xuất mảng.
b. Sắp xếp sao cho các vị trí chứa trị chẵn trên mảng vẫn chứa trị chẵn nhưng có thứ tự tăng, các vị trí chứa trị lẻ trên mảng vẫn chứa trị lẻ nhưng có thứ tự giảm.

```
Nhap n [1, 99]: 10 ↵
72 -8 45 -97 77 25 -86 86 -2 60
-86 -8 77 45 25 -97 -2 60 72 86
```

Bài giải: xem trang 122

Bài 68: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ngẫu nhiên mảng một chiều n phần tử nguyên (n chẵn) có giá trị chứa trong đoạn `[100, 200]` và xuất mảng.
b. Chia các phần tử của mảng thành hai nhóm, sao cho hiệu của tổng các phần tử nhóm này và tổng các phần tử nhóm kia là một số dương nhỏ nhất.

Tìm cặp a_0, b_0 ($a_0 > b_0$) có hiệu nhỏ nhất, cặp a_1, b_1 ($a_1 > b_1$) có hiệu nhỏ thứ hai, … Như vậy hiệu ($a_0 + a_1 + \ldots$) - ($b_0 + b_1 + \ldots$) sẽ nhỏ nhất.
Tham khảo thêm bài 62, trang 19.

```
Nhap n (n chan): 10 ↵
109 111 162 107 115 111 108 173 108 113
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
111 108 109 115 173 : 616
111 108 107 113 162 : 601
Hieu nho nhat = 15
```

Bài giải: xem trang 124

Bài 69: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ngẫu nhiên mảng một chiều `n` phần tử nguyên có giá trị chứa trong đoạn `[-100, 100]` và xuất mảng.

b. Xuất ra màn hình "run" tăng dài nhất tìm thấy đầu tiên.

"run" là chuỗi các phần tử (liên tục) theo cùng một quy luật nào đó (tăng dần, giảm dần, chẵn, lẻ, bằng nhau, …).

```
Nhap n [1, 99]: 10 ↵
-53 -32 23 78 61 -1 95 83 -55 -7
"run" tang dai nhat: -53 -32 23 78
```

Bài giải: xem trang 126

Bài 70: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ngẫu nhiên mảng một chiều `n` phần tử nguyên có giá trị chứa trong đoạn `[-100, 100]` và xuất mảng.

b. Hãy chuyển các phần tử có trị lẻ về đầu mảng, các phần tử có trị chẵn về cuối mảng. Các phần tử có trị `0` nằm ở giữa.

```
Nhap n [1, 99]: 10
-66 64 0 -50 58 51 0 45 1 82
51 45 1 0 0 -50 58 -66 64 82
```

Bài giải: xem trang 127

Bài 71: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo mảng một chiều `n` phần tử nguyên có giá trị nhập vào từ bàn phím.

b. Kiểm tra xem mảng có đối xứng hay không.

c. Hãy dịch trái xoay vòng mảng `k` lần, `k` nhập từ bàn phím.

```
Nhap n [1, 99]: 10 ↵
Nhap 10 phan tu:
1 2 3 4 5 5 4 3 2 1 ↵
Doi xung
Nhap so lan can dich: 3 ↵
4 5 5 4 3 2 1 1 2 3
```

Bài giải: xem trang 128

Bài 72: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ngẫu nhiên mảng một chiều `n` phần tử nguyên có giá trị chứa trong đoạn `[-100, 100]` và xuất mảng.

b. Kiểm tra trong mảng có số lẻ hay không? Nếu có tìm số lẻ lớn nhất.

c. Hãy dịch phải xoay vòng mảng `k` lần, `k` nhập từ bàn phím.

```
Nhap n [1, 99]: 10 ↵
4 -33 36 -4 12 72 -9 -87 76 -40
Phan tu le lon nhat: a[6] = -9
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
Nhap so lan can dich: 3 ↵
-87 76 -40 4 -33 36 -4 12 72 -9
```

Bài giải: xem trang 129

Bài 73: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo mảng một chiều `n` phần tử nguyên có giá trị nhập vào từ bàn phím.
b. In ra các phần tử trong mảng có trị phân biệt.

```
Nhap n [1, 99]: 10 ↵
Nhap 10 phan tu:
1 2 2 3 4 3 1 5 5 4 ↵
1 2 3 4 5
```

Bài giải: xem trang 131

Bài 74: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo mảng một chiều `n` phần tử nguyên có giá trị nhập vào từ bàn phím.
b. Thống kê số lần xuất hiện của các phần tử trong mảng.

```
Nhap n [1, 99]: 10 ↵
Nhap 10 phan tu:
1 2 2 3 4 3 2 5 5 3 ↵
1[1] 2[3] 3[3] 4[1] 5[2]
```

Bài giải: xem trang 132

Bài 75: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo mảng một chiều `n` phần tử nguyên có giá trị nhập vào từ bàn phím.
b. Xuất phần tử xuất hiện nhiều nhất, xuất hiện ít nhất tìm thấy đầu tiên.

```
Nhap n [1, 99]: 10 ↵
Nhap 10 phan tu:
3 2 2 3 4 3 2 5 5 3 ↵
Phan tu xuat hien nhieu nhat: 3[4]
Phan tu xuat hien it nhat: 4[1]
```

Bài giải: xem trang 133

Bài 76: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo mảng một chiều `n` phần tử nguyên có giá trị nhập vào từ bàn phím.
b. Tìm các phần tử có số lần xuất hiện là như nhau và nhiều nhất.

```
Nhap n [1, 99]: 10 ↵
Nhap 10 phan tu:
1 2 2 3 4 4 2 5 5 4 ↵
Phan tu xuat hien nhieu nhat:
2[3] 4[3]
```

Bài giải: xem trang 134

Bài 77: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ngẫu nhiên mảng một chiều `n` phần tử nguyên có giá trị chứa trong đoạn `[-100, 100]` và xuất mảng.
b. Tính tổng nghịch đảo các phần tử trong mảng.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

c. Viết hàm duyệt các phần tử `A[i]` của mảng theo thứ tự từ trái sang phải; nếu `A[i]` lẻ thì xóa một phần tử bên phải nó.

```
Nhap n [1, 99]: 10 ↵
-1 -39 62 -48 -12 -32 -39 87 75 -53
Tong nghich dao: -1
-1 62 -48 -12 -32 -39 75
```

Bài giải: xem trang 135

Bài 78: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ngẫu nhiên mảng một chiều `n` phần tử nguyên có giá trị chứa trong đoạn `[-100, 100]` và xuất mảng.

b. Hãy sắp xếp các phần tử trong mảng theo thứ tự tăng dần.

c. Hãy chèn một phần tử `x` vào trong mảng đã được sắp tăng dần mà vẫn giữ nguyên tính tăng dần của nó.

```
Nhap n [1, 99]: 10 ↵
-94 63 -78 2 7 -26 -82 8 -18 39
Mang sap xep tang:
-94 -82 -78 -26 -18 2 7 8 39 63
Nhap x: 0 ↵
-94 -82 -78 -26 -18 0 2 7 8 39 63
```

Bài giải: xem trang 136

Bài 79: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ngẫu nhiên mảng một chiều `n` phần tử nguyên có giá trị chứa trong đoạn `[-100, 100]` và xuất mảng.

b. Nhập số nguyên `x`, tìm phần tử trong mảng gần `x` nhất.

c. Viết hàm chèn `1` vào bên phải các phần tử có trị âm của mảng.

```
Nhap n (n > 0): 10 ↵
55 98 87 93 -37 -50 77 -48 93 52
Nhap x: 50
So gan x nhat: 52 ↵
55 98 87 93 -37 1 -50 1 77 -48 1 93 52
```

Bài giải: xem trang 138

Bài 80: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo mảng một chiều `A`, `n` phần tử nguyên có giá trị nhập vào từ bàn phím.

b. Tạo mảng một chiều `B`, `m` phần tử nguyên (`m` ≤ `n`), có giá trị nhập vào từ bàn phím. Tìm vị trí xuất hiện đầu tiên của mảng `B` trong mảng `A`.

c. Tìm số nguyên âm cuối cùng của mảng `A`.

```
Nhap n [1, 99]: 10 ↵
Nhap 10 phan tu mang A:
-1 2 -3 4 -5 5 -4 3 -2 1 ↵
Nhap m [1, 10]: 4 ↵
Nhap 4 phan tu mang B:
4 -5 5 -4 ↵
B co trong A tai: A[3]
So nguyen am cuoi: -2
```

Bài giải: xem trang 140

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 81: Viết hàm trộn hai mảng A, B đã được sắp xếp tăng, sao cho mảng kết quả là một mảng sắp xếp giảm. Không được sắp xếp trực tiếp mảng kết quả.

```
Nhap so phan tu mang A va B (n, m > 0): 5 7 ↵
17 -21 0 100 -42
67 11 -66 32 52 22 -48
Mang A sap tang: -42 -21 0 17 100
Mang B sap tang: -66 -48 11 22 32 52 67
Tron A va B thanh C sap giam:
100 67 52 32 22 17 11 0 -21 -42 -48 -66
```

Bài giải: xem trang 141

Bài 82: Viết chương trình cho phép người dùng nhập n số tùy ý, nhập cho đến khi nhấn Ctrl+Z. Hãy lưu các số này thành một tập hợp chứa các phần tử có trị phân biệt.

```
Nhap khong qua 100 phan tu (Ctrl+Z de dung)
1 3 5 7 2 4 3 6 7 5 4 8 ↵
^Z ↵
Tap hop A: {1, 3, 5, 7, 2, 4, 6, 8}
```

Bài giải: xem trang 143

Bài 83: Cho hai mảng A, B là hai tập hợp, khởi tạo trước hoặc nhập từ bàn phím. Tạo mảng C là một tập hợp gồm các phần tử:

a. Xuất hiện trong cả A và B (giao).
b. Không xuất hiện trong B (hiệu).
c. Xuất hiện trong A hoặc B (hợp).

A ∩ B A ∪ B A \ B

```
Tap hop A: {1, 2, 3, 5}
Tap hop B: {1, 3, 6, 7}
C = A * B: {1, 3}
C = A + B: {1, 2, 3, 5, 6, 7}
C = A \ B: {2, 5}
```

Bài giải: xem trang 144

Bài 84: Viết chương trình nhập thực hiện những yêu cầu sau:

a. Viết hàm chèn từng phần tử vào một mảng số nguyên sao cho mảng luôn giữ thứ tự giảm.
b. Dùng hàm này để lưu các trị nhập thành một mảng có thứ tự giảm. Nhập cho đến khi nhấn Ctrl+Z, xuất mảng để kiểm tra.

```
Nhap khong qua 100 phan tu (Ctrl+Z de dung):
3 5 4 7 2 6 9 1 8 ↵
^Z ↵
9 8 7 6 5 4 3 2 1
```

Bài giải: xem trang 145

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

MẢNG CỦA CÁC MẢNG[11]

Bài 85: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ngẫu nhiên mảng hai chiều vuông có kích thước `4 x 4` với các phần tử ngẫu nhiên trong đoạn `[-100, 100]` và xuất mảng.

b. Sắp xếp lại các phần tử của mảng hai chiều trên sao cho mỗi dòng tăng từ trái sang phải và mỗi cột tăng từ trên xuống dưới.

```
Mang goc:
  100   21  -67  -81
    1   55   31  -61
   48   99   62    3
   51   27  -61   14
Mang sau khi sap xep:
  -81  -67  -61  -61
    1    3   14   21
   27   31   48   51
   55   62   99  100
```

Bài giải: xem trang 146

Bài 86: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ngẫu nhiên mảng hai chiều vuông có kích thước `n x n` (`n` nhập từ bàn phím) với các phần tử ngẫu nhiên trong đoạn `[-100, 100]` và xuất mảng.

b. Sắp xếp lại các cột của mảng hai chiều trên sao cho tổng trị các phần tử của mỗi cột tăng dần từ trái sang phải.

```
Nhap n (n < 20): 3 ↵
  -48   61   31
  -22  -23  -62
   83  -24  -18
Mang sau khi sap xep:
   61   31  -48
  -23  -62  -22
  -24  -18   83
```

Bài giải: xem trang 148

Bài 87: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ma trận `A` vuông bậc `n` (`n` nhập từ bàn phím) với các phần tử được nhập từ bàn phím, xuất ma trận.

b. Tính tổng các phần tử trên đường chéo chính (vết - trace) của ma trận `A`.

c. Kiểm tra xem ma trận `A` có phải là ma trận tam giác trên hay không. Nếu phải, tính định thức của ma trận đó.

Ma trận tam giác trên là ma trận có các phần tử nằm dưới đường chéo chính đều bằng `0`. Định thức của ma trận tam giác trên theo Gauss[12]:

$$\det(A) = \prod_{i=1}^{n} A_{ii}$$

[11] C không hỗ trợ mảng nhiều chiều (ví dụ a[5,7]), C chỉ hỗ trợ mảng có kiểu bất kỳ, kể cả mảng *chứa các mảng* (ví dụ a[5][7]). Để dễ trình bày, chúng tôi vẫn dùng thuật ngữ mảng 1 chiều, 2 chiều, …

[12] Carl Friedrich Gauss (1777 - 1855)

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
Nhap bac ma tran: 3 ↵
a[0][0] = 1 ↵
...
a[2][2] = 9 ↵
    1    2    3
    0    5    6
    0    0    9
Trace = 15
det(A) = 45
```

Bài giải: xem trang 149

Bài 88: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ma trận vuông bậc `n` (`n` nhập từ bàn phím) với các phần tử là các trị ngẫu nhiên trong đoạn `[-100, 100]`, xuất ma trận.

b. Viết hàm kiểm tra xem ma trận có đồng nhất hay không. Nếu không, in ra ma trận đồng nhất cùng bậc với ma trận trên.

> Ma trận đồng nhất I_n (identity matrix) là ma trận vuông bậc `n` có các phần tử đều bằng `0`, trừ các phần tử trên đường chéo chính đều bằng `1`.

```
Nhap bac ma tran: 3 ↵
   43   65   29
   78  -67   26
  -72  -61   95
Ma tran tren khong dong nhat
    1    0    0
    0    1    0
    0    0    1
```

Bài giải: xem trang 150

Bài 89: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ma trận vuông bậc `n` (`n` nhập từ bàn phím) với các phần tử là các trị ngẫu nhiên trong đoạn `[-100, 100]`, xuất ma trận.

b. Hoán chuyển phần tử lớn nhất nằm trên mỗi dòng với phần tử nằm trên đường chéo chính cũng thuộc dòng đó.

```
Nhap bac ma tran: 3 ↵
  -27   11  -81
  -10  -13   35
  -24   61  -17
Ma tran sau khi sap xep:
   11  -27  -81
  -10   35  -13
  -24  -17   61
```

Bài giải: xem trang 151

Bài 90: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ma trận vuông bậc `n` (`n` nhập từ bàn phím) với các phần tử là các trị ngẫu nhiên trong đoạn `[-100, 100]`, xuất ma trận.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

b. Viết hàm xuất tất cả các đường chéo song song với đường chéo phụ, mỗi đường chéo trên một dòng.

```
Nhap bac ma tran: 3 ↵
  -84   50   68
   53  -94  -47
  -62  -59   67
Cac duong cheo song song cheo phu:
  -84
   50   53
   68  -94  -62
  -47  -59
   67
```

Bài giải: xem trang 152

Bài 91: Một mẫu lát (pattern) thường được định nghĩa dưới dạng một ma trận nhị phân (xem hình dưới). Giả sử mẫu lát trên dùng lát kín ma trận `A` (kích thước `20 x 20`), xuất ma trận `B` là ma trận con của `A`, kích thước `8 x 12`, có góc trên trái nằm tại phần tử `A[3][4]`.

```
 * *
*   *
*   *
 * *
```

```
010100
100010
100010
010100
```

```
   * *   * *
   * *   * *
* *   * *
* *   * *
   * *   * *
   * *   * *
* *   * *
* *   * *
```

Bài giải: xem trang 153

Bài 92: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ma trận vuông bậc `n` (`n` nhập từ bàn phím) với các phần tử là các trị ngẫu nhiên trong đoạn `[-100, 100]`, xuất ma trận.

b. Tìm đường chéo, song song với đường chéo chính và có tổng trị các phần tử là lớn nhất.

```
Nhap bac ma tran: 3 ↵
  -99  -24  -89
   97  -66   10
   16   22  -66
Duong cheo co tong lon nhat:
97 22 : 119
```

Bài giải: xem trang 154

Bài 93: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ma trận `A` vuông bậc 3 với các phần tử là các trị ngẫu nhiên trong đoạn `[0, 9]`, xuất ma trận.

b. Với `k` nguyên dương nhập từ bàn phím (`k > 1`), xuất ma trận lũy thừa A^k.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Nếu A là một ma trận n x n và nếu k là một số nguyên dương thì A^k là ký hiệu của tích k lần ma trận A, gọi là ma trận lũy thừa A^k. $A^0 = I$ (ma trận đồng nhất). Ma trận lũy thừa thường dùng khi xác định tính liên thông của đồ thị bằng ma trận kề. Các phần tử của ma trận tích C = A.B là $C_{ij} = \sum_{k=1}^{n} A_{ik}B_{kj}$. Tích A.B tồn tại khi số cột của A bằng số dòng của B.

```
    3    8    8
    9    2    8
    9    3    6
Nhap luy thua (k > 1): 3 ↵
 2259 1760 2552
 2475 1544 2552
 2376 1452 2424
```

Bài giải: xem trang 155

Bài 94: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ma trận A vuông bậc n (n > 1, nhập từ bàn phím) với các phần tử là các trị ngẫu nhiên trong đoạn [-100, 100], xuất ma trận.

b. Tạo rồi xuất ma trận con B vuông bậc n - 1 từ ma trận A bằng cách loại bỏ dòng và cột chứa phần tử có trị tuyệt đối lớn nhất.

```
Nhap bac ma tran (n > 1): 3 ↵
  -56   42    5
  -18   82   18
   50  -39   42
Ma tran B:
  -56    5
   50   42
```

Bài giải: xem trang 157

Bài 95: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ma trận A vuông bậc n (n nhập từ bàn phím) với các phần tử là các trị ngẫu nhiên trong đoạn [-100, 100], xuất ma trận.

b. Cho k nguyên với 0 ≤ k < n. Tạo và xuất ma trận B vuông bậc n từ ma trận A bằng cách hoán chuyển dòng k với cột k.

```
Nhap bac ma tran: 3 ↵
  -52   31  -75
  -92   59   27
  -58   47   29
Nhap k: 1 ↵
  -52  -92  -75
   31   59   47
  -58   27   29
```

Bài giải: xem trang 159

Bài 96: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ma trận A vuông bậc n (n nhập từ bàn phím) với các phần tử có trị: A_{ij} = sin(i - 2 * j / π), xuất ma trận.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

b. Đếm số phần tử không âm của ma trận.

```
Nhap bac ma tran: 4 ↵
  0.000000 -0.594481 -0.956056 -0.943067
  0.841471  0.355436 -0.269852 -0.789417
  0.909297  0.978566  0.664452  0.090019
  0.141120  0.702007  0.987862  0.886692
Co 11 phan tu khong am
```

Bài giải: xem trang 159

Bài 97: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ma trận `A` có kích thước `n x m` (`n`, `m` nhập từ bàn phím) với các phần tử là các trị ngẫu nhiên trong đoạn `[-100, 100]`, xuất ma trận.

b. Lân cận của phần tử A_{ij} được hiểu là các phần tử có chỉ số tương ứng chênh lệch với `i`, `j` không quá `1` đơn vị. Xuất ma trận nhị phân `B` (gọi là ma trận cực tiểu) có kích thước `m x n` với B_{ij} `= 1` khi tất cả những phần tử lân cận A_{ij} đều lớn hơn A_{ij} (khi đó A_{ij} được gọi là phần tử cực tiểu), các phần tử B_{ij} còn lại bằng `0`.

```
Nhap n, m: 3 4 ↵
   14   78   54  -43
   84   71  -45   22
  -54   35   14   83
Ma tran cuc tieu:
    1    0    0    0
    0    0    1    0
    1    0    0    0
```

Bài giải: xem trang 160

Bài 98: Viết chương trình thực hiện những yêu cầu sau:

c. Tạo ma trận có kích thước `n x m` (`n`, `m` nhập từ bàn phím) với các phần tử là các trị ngẫu nhiên trong đoạn `[-100, 100]`, xuất ma trận.

d. In ma trận xoay `90°` theo chiều kim đồng hồ từ của ma trận trên. Tâm quay là phần tử góc dưới trái.

```
Nhap n, m: 3 4 ↵
   63  -49   88   95
  -52  -94  -33   48
   73  -69   86  -38
Ma tran sau khi quay:
   73  -52   63
  -69  -94  -49
   86  -33   88
  -38   48   95
```

Bài giải: xem trang 161

Bài 99: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo hai ma trận vuông cùng bậc `n` (`n` nhập từ bàn phím) với các phần tử là các trị ngẫu nhiên trong đoạn `[-10, 10]`, xuất hai ma trận.

b. In ma trận tổng và ma trận tích của hai ma trận trên.

```
Nhap bac ma tran: 2 ↵
Ma tran A:
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  -10    9
    7    9
Ma tran B:
    3   -1
    7    4
Ma tran tong:
   -7    8
   14   13
Ma tran tich:
   33   46
   84   29
```

Bài giải: xem trang 163

Bài 100: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ma trận có kích thước `n x m` (`n`, `m` nhập từ bàn phím) với các phần tử được nhập từ bàn phím, xuất ma trận.

b. Ma trận gọi là "thưa" (sparse matrix) nếu số phần tử có trị `0` nhiều hơn số phần tử có trị khác `0`. Kiểm tra xem ma trận trên có "thưa" hay không.

```
Nhap n, m: 3 4 ↵
a[0][0] = 3 ↵
...
a[2][3] = 0 ↵
    3    0    2   -5
    0   -4    0    0
    0    0    0    0
Ma tran thua
```

Bài giải: xem trang 165

Bài 101: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ma trận vuông bậc `n` (`n` nhập từ bàn phím) với các phần tử nhập vào từ bàn phím, xuất ma trận.

b. Xuất tổng các hàng và tổng các cột.

c. Kiểm tra xem ma trận trên có tạo thành một ma phương không.

Ma phương (magic square) là hình vuông có tổng đường chéo chính, đường chéo phụ, hàng bất kỳ, cột bất kỳ đều bằng nhau. Ví dụ, hình vuông bên phải là ma phương bậc 3 với tổng 15.

2	9	4
7	5	3
6	1	8

```
Nhap bac ma tran: 3 ↵
A[0][0] = 2 ↵
...
A[2][2] = 8 ↵
    2    9    4   Tong hang 0:  15   Tong cot 0:  15
    7    5    3   Tong hang 1:  15   Tong cot 1:  15
    6    1    8   Tong hang 2:  15   Tong cot 2:  15
Ma phuong
```

Bài giải: xem trang 166

Bài 102: Viết chương trình thực hiện những yêu cầu sau:

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

a. Tạo ma trận có kích thước `n x m` (`n`, `m` nhập từ bàn phím) với các phần tử là các trị ngẫu nhiên trong đoạn `[-100, 100]`, xuất ma trận.
b. Chuyển ma trận `A` thành ma trận chuyển vị `A*` và in ra.

Ma trận A^T kích thước `m x n` với $A^T_{ji} = A_{ij}$ được gọi là ma trận chuyển vị (transpose) của ma trận `A` kích thước `n x m`.

```
Nhap n, m: 3 4 ↵
  -31   79  -30   43
   60  -69   87  -36
   72   10  -63   53
Ma tran chuyen vi:
  -31   60   72
   79  -69   10
  -30   87  -63
   43  -36   53
```

Bài giải: xem trang 167

Bài 103: Viết chương trình thực hiện những yêu cầu sau:
a. Tạo ma trận vuông bậc `n` (`n` nhập từ bàn phím) với các phần tử là các trị ngẫu nhiên trong đoạn `[-10, 10]`, xuất ma trận.
b. Xuất ma trận `B` từ ma trận nguồn `A`, sao cho `B[i][j]` bằng tổng các phần tử thuộc hai đường chéo đi qua phần tử `A[i][j]`.

```
Nhap bac ma tran: 3 ↵
   -4   -6   -7
    7    8   -9
    4    7  -10
Ma tran B:
   -6   -8    5
    8   -9   -8
    5    5   -6
```

Bài giải: xem trang 168

Bài 104: Viết chương trình thực hiện những yêu cầu sau:
a. Tạo ma trận vuông bậc `n` (`n` nhập từ bàn phím) với các phần tử là các trị ngẫu nhiên trong đoạn `[-10, 10]`, xuất ma trận.
c. Xuất ma trận `B` từ ma trận nguồn `A`, sao cho `B[i][j]` bằng số lớn nhất trong hai đường chéo đi qua phần tử `A[i][j]`.

```
Nhap bac ma tran: 3 ↵
   -3    3   -1
    1   -7   -4
    2   -5   -7
Ma tran B:
   -3    3    2
    3    2    3
    2    1   -3
```

Bài giải: xem trang 170

Bài 105: Viết chương trình thực hiện những yêu cầu sau:

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

a. Tạo ma trận có kích thước `n x m` (`n`, `m` nhập từ bàn phím) với các phần tử là các trị ngẫu nhiên trong đoạn `[-10, 10]`, xuất ma trận.
b. Tạo ma trận `B` từ ma trận nguồn `A` sao cho `B[i][j]` bằng tổng các phần tử không thuộc dòng `i` cột `j` của ma trận `A`.

```
Nhap bac ma tran: 3 ↵
    9    3    0
  -10   -8   -4
   -1    0    1
Ma tran B:
  -11  -14  -19
    4    9   11
   -9   -5   -6
```

Bài giải: xem trang 170

Bài 106: Viết chương trình thực hiện những yêu cầu sau:
a. Tạo ma trận vuông bậc `n` (`n` nhập từ bàn phím) với các phần tử là các trị ngẫu nhiên trong đoạn `[-10, 10]`, xuất ma trận.
b. Tìm các điểm yên ngựa của ma trận trên.

> Phần tử A_{ij} gọi là *điểm yên ngựa* (saddle point) của ma trận `A` nếu nó vừa là phần tử nhỏ nhất của dòng `i`, đồng thời là phần tử lớn nhất của cột `j`; hoặc ngược lại, vừa là phần tử lớn nhất của dòng `i`, đồng thời là phần tử nhỏ nhất của cột `j`.
> Một ma trận có thể *không có* hoặc có nhiều điểm yên ngựa.

```
Nhap bac ma tran: 3 ↵
   -2    5   -1
   -3    1   -2
   -6    2   -4
MIN dong MAX cot: a[0][0] = -2
MAX dong MIN cot: a[1][1] = 1
```

Bài giải: xem trang 171

Bài 107: Tạo ma trận vuông bậc 5 với các phần tử có trị từ 1, 2, …, 25, sắp xếp giảm theo hình zigzag từ trái sang phải, từ trên xuống dưới.

```
  1  2  3  4  5
 10  9  8  7  6
 11 12 13 14 15
 20 19 18 17 16
 21 22 23 24 25
```

Bài giải: xem trang 172

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 108: Tạo ma trận vuông bậc `n` (`n` nhập từ bàn phím, `n < 20`) với các phần tử có trị từ 1 đến n^2 sắp xếp tăng theo hình xoắn ốc từ trái sang phải, từ ngoài vào trong.

```
Nhap bac ma tran (n < 20): 5 ↵
  1  2  3  4  5
 16 17 18 19  6
 15 24 25 20  7
 14 23 22 21  8
 13 12 11 10  9
```

Bài giải: xem trang 173

Bài 109: Tạo ma trận vuông bậc 5 với các phần tử có trị từ 1, 2, …, 25, xuất ma trận. Viết chương trình sắp xếp trị của các phần tử trong ma trận tăng theo thứ tự như hình dưới:

Thực chất là ghi lần lượt các phần tử lên các đường chéo song song với đường chéo phụ. Mỗi lần chuyển sang đường chéo mới thì lại đổi chiều ghi, thực chất là chuyển vị các phần tử đang ghi.

```
  1  2  6  7 15
  3  5  8 14 16
  4  9 13 17 22
 10 12 18 21 23
 11 19 20 24 25
```

Bài giải: xem trang 174

Bài 110: Viết chương trình thực hiện những yêu cầu sau:

a. Cấp phát động ma trận có kích thước `n x m` (`n`, `m` nhập từ bàn phím) với các phần tử là các trị ngẫu nhiên trong đoạn `[-10, 10]`, xuất ma trận.

b. Viết hàm `int sumNeg( int a[], int m )` trả về số các số âm có trong mảng một chiều `a`, kích thước `m`. Sử dụng hàm này để tìm số các số âm của dòng `k` (`0 ≤ k < m`, nhập từ bàn phím) trong ma trận trên.

```
Nhap n, m: 5 4 ↵
   -7   -3   -4    3
    3    5    5    9
    7   -6    9    3
    9    4    5    7
    5    3   -2    6
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
Nhap dong k: 2
Dong 2 co 1 so am
```

Bài giải: xem trang 174

Bài 111: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo ma trận có kích thước `n x m` (`n`, `m` nhập từ bàn phím) với các phần tử là các trị ngẫu nhiên trong đoạn `[-10, 10]`, xuất ma trận.

b. Hãy dịch trái xoay vòng các phần tử trong ma trận theo hình tròn xoắn ốc từ trái sang phải, từ ngoài vào trong `k` bước (`k` nhập từ bàn phím).

```
Nhap dong, cot: 4 6 ↵
    6   -1   -1    8   -8   -1
    0   -8    9    5   -3   -3
  -10    3   -5   -2   -2    7
    5  -10   -1    0    4   10
Nhap buoc dich: 2 ↵
   -5    3    6   -1   -1    8
    5  -10    0   -8    9   -8
  -10   -2   -2   -3    5   -1
   -1    0    4   10    7   -3
```

Bài giải: xem trang 176

Bài 112: Cho ma trận vuông bậc `n` (nhập từ bàn phím), chứa các phần tử là các số thực ngẫu nhiên. Thực hiện các phép biến đổi sơ cấp hàng trên ma trận (phép khử Gauss) để đưa ma trận trở thành ma trận tam giác trên.

Định lý về các phép biến đổi sơ cấp:
- Định thức trên ma trận không đổi nếu thay một hàng trên ma trận bởi một hàng mới bằng hiệu của hàng cũ với một hàng khác của ma trận nhân với một hệ số `k` bất kỳ.
- Định thức của ma trận đổi dấu nếu hoán vị hai hàng trên ma trận.

Phép khử Gauss: với mỗi phần tử A_{ii} trên đường chéo chính của ma trận:
- Nếu A_{ii} khác `0`, khử các phần tử trên cột `i` nằm dưới A_{ii} bằng cách thay các hàng A_{jk} bởi hiệu: $A_{jk} - A_{ik} * A_{ji}/A_{ii}$
với `j = (i+1)..(n-1)` và `k = i..(n-1)`.
- Nếu A_{ii} bằng `0`, tìm phần tử A_{ji} đầu tiên `0` khác với `j = (i+1)..(n-1)`; nếu tìm thấy, hoán vị hàng `i` với hàng `j` (và đổi dấu định thức nếu cần tính); nếu không tìm thấy, định thức bằng `0`.

```
Nhap bac cua ma tran: 3 ↵
   0.943   0.217   0.075
   0.304   0.165   0.564
   0.047   0.178   0.682

   0.943   0.217   0.075
   0.000   0.095   0.540
   0.000   0.000  -0.272
```

Bài giải: xem trang 180

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

CHUỖI

Bài 113: Nhập một vào chuỗi số nhị phân. Tìm trong đó chuỗi con dài nhất chứa toàn các số `0`.

```
Nhap chuoi nhi phan: 100101000010001001001 ↵
Chuoi 0 dai nhat co 4 ky tu
Bat dau tai s[6]
```

Bài giải: xem trang 181

Bài 114: Viết các hàm mô phỏng các hàm của `string.h` sau: `strlen()`, `strcpy()`, `strcat()`. Thử nghiệm các hàm này.

```
Chuoi 1: [the quick brown fox ] (20)
Chuoi 2: [jumps over the lazy dog] (23)
strcpy( buf, s1 ) roi strcat( buf, s2 ):
[the quick brown fox jumps over the lazy dog] (43)
```

Bài giải: xem trang 182

Bài 115: Viết các hàm mô phỏng các hàm của `string.h` sau: `strcmp()`, `strchr()`, `strrchr()`. Thử nghiệm các hàm này.

```
Chuoi goc s: [jackdaws love my big sphinx of quartz]
strchr( s, 'm' ) : [my big sphinx of quartz]
strrchr( s, 'o' ): [of quartz]
Sap xep cac chuoi dung strcmp():
black  blue  brown
```

Bài giải: xem trang 184

Bài 116: Viết các hàm mô phỏng của các hàm `string.h` sau: `strspn()`, `strncmp()`, `strstr()`. Thử nghiệm các hàm này.

```
Chuoi kiem tra : hom qua qua noi qua ma qua khong qua
Vi tri tu 'qua':     x   x       x      x         x
Ky tu dau tien cua s = 'cabbage' khong co trong 'abc' la s[5]
```

Bài giải: xem trang 186

Bài 117: Viết các hàm:

a. Mô phỏng hàm `LEFT` (trả về `n` ký tự bên trái chuỗi) của Microsoft Excel.
b. Mô phỏng hàm `RIGHT` (trả về `n` ký tự bên phải chuỗi) của Microsoft Excel.

```
Chuoi goc: [Kernighan and Ritchie]
left( str, 9 ) : [Kernighan]
right( str, 7 ): [Ritchie]
```

Bài giải: xem trang 189

Bài 118: Viết chương trình nhập vào một chuỗi, ngoài các ký tự chữ cái còn cho phép các ký tự space, tab, nháy đơn, nháy đôi, chấm hỏi, chấm than, chấm, phẩy. Xử lý chuỗi như sau:

a. Bỏ các ký tự space thừa (các ký tự space đầu chuỗi, cuối chuỗi, giữa các từ chỉ chừa lại một ký tự space).
b. Xuất các từ phân biệt, có viết hoa các ký tự đầu mỗi từ, viết thường các ký tự còn lại của từ.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
Chuoi goc      : [  'bJARne?  sTROUstRUP'   ]
Loai space du: ['bJARne? sTROUstRUP']
Cac tu da chuan hoa:
   Bjarne, Stroustrup
```

Bài giải: xem trang 190

Bài 119: Viết chương trình nhập vào một chuỗi tối đa 255 ký tự.

a. Đếm số từ có trong chuỗi biết giữa các từ có ít nhất một ký tự space, các dấu "." và ",", không có ký tự đặc biệt khác.

b. Thống kê tần số xuất hiện các từ có 1, 2, 3, 4, 5, 6, 7 ký tự có trong chuỗi nhập trên.

```
Nhap chuoi: Chieu nay, tren song, khong mot con do. ↵
Co 8 tu
Tan so xuat hien cac tu:
1[0]  2[1]  3[3]  4[2]  5[2]  6[0]  7[0]
```

Bài giải: xem trang 191

Bài 120: Viết chương trình yêu cầu nhập vào hai chuỗi ký tự, mỗi chuỗi có chiều dài tối đa 80 ký tự. Chèn chuỗi 2 vào chuỗi 1 tại vị trí thứ `k` (`k` nhập từ bàn phím, với `0` ≤ `k` ≤ chiều dài chuỗi 1).

```
Chuoi goc  : Em noi sao, roi sao? ↵
Chuoi chen :  doi tre muon ↵
Vi tri chen: 11 ↵
Chuoi ket qua: Em noi sao, doi tre muon roi sao?
```

Bài giải: xem trang 192

Bài 121: Viết chương trình yêu cầu nhập vào chuỗi ký tự có chiều dài tối đa là 80 ký tự. Nhập vào hai số nguyên dương `n` và `p`, trong chuỗi trên tiến hành xóa `n` ký tự bắt đầu từ vị trí `p`.

```
Nhap chuoi: Con dieu roi cho vuc tham buon theo ↵
Nhap vi tri dau: 9 ↵
Nhap so ky tu loai bo: 17 ↵
Chuoi ket qua: Con dieu buon theo
```

Bài giải: xem trang 193

Bài 122: Viết chương trình nhập vào một số nhị phân ở dạng chuỗi ký tự `0` và `1`, chuyển số này thành một số nguyên hệ thập phân.

```
Nhap chuoi nhi phan: 10011011110101001010001 1 ↵
Tri thap phan: 10212515
```

Bài giải: xem trang 194

Bài 123: Viết chương trình chỉ dùng các thao tác trên chuỗi, cộng hai số nhị phân ở dạng chuỗi ký tự, in ra kết quả cũng ở dạng chuỗi ký tự.

```
    1101010000110001
      11000000111001
   -----------------
   10000010001101010
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài giải: xem trang 195

Bài 124: Viết hàm `itos()` nhận vào một số nguyên dương, chuyển số nguyên dương này thành chuỗi ký tự. Viết hàm `stoi()` ngược lại, nhận vào một chuỗi ký tự số mô tả một số nguyên dương, chuyển chuỗi này thành số nguyên dương.

```
Nhap so nguyen duong n: 12345 ↵
itos() -> 12345
stoi() -> 12345
```

Bài giải: xem trang 197

Bài 125: Viết chương trình nhập vào hai số nguyên dương rất lớn lưu dạng chuỗi ký tự, in ra kết quả nhân hai số nguyên dương trên cũng ở dạng chuỗi ký tự. Trình bày giống như bài toán nhân tay. Thực hiện hoàn toàn bằng các thao tác trên chuỗi.

```
So bi nhan: 87654321 ↵
So nhan   : 12345678 ↵
            87654321
  *
            12345678
--------------------
           701234568
          613580247
         525925926
        438271605
       350617284
      262962963
     175308642
     87654321
--------------------
    1082152022374638
```

Bài giải: xem trang 198

Bài 126: Viết chương trình nhập một chuỗi và một ký tự. Tìm tất cả các vị trí xuất hiện của ký tự `c` trong một chuỗi `s`.

```
Nhap chuoi: Mai sau khoc nhung roi don dinh menh ↵
Tìm ky tu nao? o
Vi tri xuat hien: 10 20 24
```

Bài giải: xem trang 200

Bài 127: Không sử dụng các hàm của `string.h`, viết chương trình đảo các từ của một chuỗi cho sẵn hoặc nhập từ bàn phím. Chuỗi có thể có những ký tự đặc biệt.

```
Chuoi goc: [ cam  khong  duoc  do  rac   ]
Chuoi dao: [   rac  do  duoc  khong  cam ]
```

Bài giải: xem trang 201

Bài 128: Nhập một chuỗi ký tự, xuất ra các từ dài nhất trong chuỗi.

```
Nhap chuoi: Troi dat rong nen tinh yeu de lac ↵
Troi[4] rong[4] tinh[4]
```

Bài giải: xem trang 203

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 129: Nhập một chuỗi ký tự chứa ít nhất 4 ký tự số. Loại bỏ một số ký tự ra khỏi chuỗi sao cho 4 ký tự số cuối cùng còn lại (theo đúng thứ tự đó) tạo nên số lớn nhất.

```
Nhap chuoi (it nhat 4 chu so): 24d5n4r05f704n652z393 ↵
So lon nhat con lai: 7693
```

Bài giải: xem trang 204

Bài 130: Viết hàm tìm kiếm trong một số chuỗi cho trước, một chuỗi so trùng với chuỗi mẫu. Cho phép chuỗi mẫu dùng ký tự đại diện (wildcard): `*`. Ký tự `*`: so trùng với *không* hoặc *nhiều* ký tự *bất kỳ* tại vị trí tương ứng trong chuỗi cho trước.

```
Danh sach cac tu: television menu editions education
Tim [*e*u*]: menu education
Tim [e*i*n]: education
Tim [*e*o*]: television editions education
```

Bài giải: xem trang 206

Bài 131: Viết chương trình nhập vào chuỗi ký tự có chiều dài tối đa 80 ký tự. Không phân biệt chữ hoa, chữ thường, thực hiện các công việc sau:

a. Kiểm tra xem chuỗi có đối xứng hay không?
b. Nhập vào một chuỗi ký tự rồi tiến hành xóa hết các ký tự giống với ký tự đó trong chuỗi trên.

```
Nhap chuoi: Stressed? No tips? Spit on desserts! ↵
Chuoi doi xung
Xoa ky tu nao? s ↵
treed? No tip? pit on deert!
```

Bài giải: xem trang 207

Bài 132: Viết chương trình nhập hai chuỗi và xác định xem chúng có được tạo ra với cùng các ký tự hay không. Ví dụ "`dear`" và "`reader`" là hai chuỗi tạo từ cùng các ký tự `a`, `d`, `e`, `r`.

```
Nhap chuoi a: dear ↵
Nhap chuoi b: reader ↵
Tao tu cung cac ky tu
```

Bài giải: xem trang 209

Bài 133: Viết hàm `hextoulong()` nhận một chuỗi mô tả số `unsigned long int` thuộc hệ thập lục phân, trả về số `unsigned long int` thuộc hệ thập phân tương ứng.

Dùng phương pháp Horner[13] như sau:

1. Đặt `x = 0`
2. Đặt `i` bằng thứ tự ký tự trái nhất của chuỗi hex, thứ tự tính từ `0`, phải sang trái. h_i là trị hex tại vị trí `i`.
3. Nhân `x` với `16` rồi cộng h_i với `x`
4. Giảm `i` đi `1`.

Ví dụ: số `0F4D`

i	h_i	x = 16 * x + h
3	0	0
2	F	16 * 0 + F = 15
1	4	16 * 15 + 4 = 244
0	D	16 * 244 + D = 3917

[13] William George Horner (1786 - 1837)

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

5. Nếu i ≥ 0, lặp lại bước 3, 4
6. Trả về x

```
Nhap chuoi hex: 3AdE68b1 ↵
Decimal: 987654321
```

Bài giải: xem trang 209

Bài 134: Viết hàm `rtoi()` nhận một số La mã dưới dạng chuỗi ký tự, trả về số nguyên tương ứng.

```
M: 1000     D: 500      C: 100
L: 50       X: 10       V: 5        I: 1
```

Ký hiệu thể hiện 10^x không được đứng ngay trước ký hiệu lớn hơn 10^{x+1}.
Ví dụ: số 99 là XCIX, không phải IC.

```
MCMXCIX = 1999
MCMIC = So khong hop le
```

Bài giải: xem trang 210

Bài 135: Viết chương trình thay thế tất cả các chuỗi con cho trước (thường gọi là các mẫu - pattern) trong một chuỗi bằng một chuỗi khác. Xuất chuỗi kết quả.

```
Chuoi goc: ta mo thay em o mot noi em xa lam
Thay the 'em' voi 'em yeu'
Chuoi moi: Ta mo thay em yeu o mot noi em yeu xa lam
```

Bài giải: xem trang 211

Bài 136: Nhập vào số dòng văn bản cho đến khi nhấn Ctrl+Z. Viết chương trình đếm số và in số dòng, số từ, tần số, các chữ cái trong các dòng văn bản đó.

```
Que nha me co gian thien ly, ↵
Mot khoang ram hien giua nang trua. ↵
Va nhung chuyen nghe hoai khong biet chan, ↵
Bat dau la: Ngay xua, ngay xua... ↵
^Z ↵
4 hang, 29 tu, voi tan so cac ky tu:
A: 17   B: 2    C: 3    D: 1    E: 7    F: 0    G: 9
H: 10   I: 6    J: 0    K: 2    L: 2    M: 3    N: 15
O: 5    P: 0    Q: 1    R: 2    S: 0    T: 5    U: 8
V: 1    W: 0    X: 2    Y: 4    Z: 0
```

Bài giải: xem trang 212

Bài 137: Viết chương trình đọc các tên đầy đủ, giữa các từ của tên chỉ có ký tự space, mỗi tên trên một dòng, cho đến khi nhấn Ctrl+Z (hoặc Ctrl+D nếu trên Unix). Sau đó in chúng ta theo dạng niên giám điện thoại chuẩn. Ví dụ:

```
Wolfgang Amadeus Mozart ⇒ Mozart, Wolfgang A.
```

```
George Frederic Handel ↵
Carl Philipp Emanuel Bach ↵
Wolfgang Amadeus Mozart ↵
^Z ↵
Handel, George F.
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
Bach, Carl P. E.
Mozart, Wolfgang A.
```

Bài giải: xem trang 213

Bài 138: Viết chương trình đọc vào một số dòng, mỗi dòng không quá 50 ký tự, cho đến khi nhấn Ctrl+Z. Sau đó in lại các dòng đó theo dạng thức canh phải với chiều ngang dòng 50 ký tự.

```
Xa em, gio it, lanh nhieu, ↵
Lua khuya tan cham, mua chieu do nhanh. ↵
(Tran Huyen Tran) ↵
^Z ↵
                        Xa em, gio it, lanh nhieu,
          Lua khuya tan cham, mua chieu do nhanh.
                                (Tran Huyen Tran)
```

Bài giải: xem trang 214

Bài 139: Nén run-length: một chuỗi ký tự (không chứa ký tự số) có thể nén bằng cách sau: Chuỗi con gồm `n` (`n > 1`) ký tự `a` giống nhau sẽ được thay thế bằng `na` (chú ý, `n` có thể có nhiều hơn `1` ký tự). Ví dụ: `aabcccd` được nén thành `2ab3cd`. Viết chương trình nén và giải nén run-length chuỗi ký tự (không chứa ký tự số) nhập vào.

```
Chuoi goc: aaabccccddddeeeeeeeeeeeeefghhhhhhiiiiaaaabbbbbbc
Nen      : 3ab4c4d12efg6h4i4a6bc [44.7%]
Giai nen : aaabccccddddeeeeeeeeeeeeefghhhhhhiiiiaaaabbbbbbc
```

Bài giải: xem trang 215

Bài 140: ISBN (International Standard Book Number, phát âm "is-ben") là mã số duy nhất cho sách xuất bản trên thế giới. ISBN bao gồm 13 ký tự (0 - 9, ký tự nối - (hyphen), X) chia thành 4 phần bởi ký tự nối: định danh nhóm (quốc gia, ngôn ngữ, …) định danh nhà xuất bản, định danh sách của nhà xuất bản đó, và số kiểm tra. Số kiểm tra được dùng kết hợp với các số khác trong một thuật toán kiểm tra số ISBN, chỉ chứa một ký tự (0 đến 9, X thay cho 10).
Viết chương trình kiểm tra một số chuỗi ISBN nhập vào có hợp lệ hay không.

Thuật toán đơn giản dùng để kiểm tra tính hợp lệ của số ISBN: lấy từng số của ISBN nhân với số thứ tự chỉ vị trí của nó (bắt đầu từ 1, tính từ phải sang trái, không tính dấu nối –). Tổng các tích nhận được nếu chia hết cho 11 thì số ISBN được kiểm tra là hợp lệ. Ví dụ:

```
ISBN   0 - 1    3   1 - 1   0    3    7   0 - 9
Vị trí 10   9    8   7   6   5    4    3   2   1
Tích    0 + 9 + 24 + 7 + 6 + 0 + 12 + 21 + 0 + 9 = 88
```

88 chia hết cho 11, vậy số ISBN trên hợp lệ.

```
ISBN 0-133-62658-x hop le
```

Bài giải: xem trang 216

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

ĐỆ QUY

Bài 141: Giả sử dân số thế giới năm 2000 là 8 tỷ người với mức tăng trưởng hàng năm không đổi là 2,5%. Viết hàm đệ quy tính dân số thế giới năm 2010.

```
10240676354 nguoi
```

Bài giải: xem trang 217

Bài 142: Viết chương trình nhập vào một dòng văn bản, kết thúc bằng phím Enter, sau đó hiển thị dòng văn bản đảo của văn bản nhập. Dùng giải thuật đệ quy với hàm `main()`, không dùng các cấu trúc lưu trữ như mảng, chuỗi.

Ngôn ngữ C cho phép đệ quy ngay cả với hàm `main()`

```
race fast safe car ↵
rac efas tsaf ecar
```

Bài giải: xem trang 218

Bài 143: Cho hàm Ackermann[14] với `n` và `m` không âm:

$$A(n,m) = \begin{cases} m+1 & n = 0 \\ A(n-1,1) & m = 0 \\ A(n-1, A(n, m-1)) & n, m > 0 \end{cases}$$

Tính `A(3, 6)`. Cho biết số lần gọi đệ quy khi tính `A(3, 6)`.

```
A( 3, 6 ) = 509
Goi de quy 172233 lan
```

Bài giải: xem trang 219

Bài 144: Viết hàm đệ quy `double Pi()` không tham số tính số `Pi` theo công thức:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + .. + (-1)^k \frac{1}{2k+1}$$ (với sai số epsilon 10^{-3}, 4/n < epsilon)

```
Pi = 3.141
```

Bài giải: xem trang 220

Bài 145: Viết các hàm đệ quy

`void ToBin( int n )` ⇒ xuất dạng nhị phân của `n` (`n` nguyên dương).

`void ToHex( int n )` ⇒ xuất dạng thập lục phân của `n` (`n` nguyên dương).

```
Nhap x: 123 ↵
Bin: 1111011
Hex: 7B
```

Bài giải: xem trang 221

Bài 146: Nhập vào số nguyên `n` `(0 < n ≤ 9)`. Xuất tam giác Pascal chiều cao `n`.

[14] Wilhelm Ackermann (1896 - 1962)

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Tam giác Pascal[15] được xác định như sau:

C_0^0

$C_1^0 \quad C_1^1$

...

$C_n^0 \quad C_n^1 \ldots C_n^n$

Trong đó $C_n^k = \frac{n!}{k!\,(n-k)!}, 0 \le k \le n$ là tổ hợp n chập k (trong trường hợp này gọi là các hệ số của nhị thức Newton[16]).

C_n^k còn được xác định bằng công thức truy hồi:

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

```
Nhap n (n < 9): 4 ↵

            1
          1   1
        1   2   1
      1   3   3   1
    1   4   6   4   1
```

Bài giải: xem trang 222

Bài 147: Viết hàm sắp xếp *tăng* một mảng theo kiểu chọn (SelectionSort) với giải thuật đệ quy: `void SelectSort( int a[], int n )` (`n` là số phần tử).

```
Mang goc : 3 5 4 6 7 1 2
Mang tang: 1 2 3 4 5 6 7
```

Bài giải: xem trang 223

Bài 148: Viết hàm sắp xếp *giảm* một mảng theo kiểu nổi bọt (BubbleSort) với giải thuật đệ quy: `void BubbleSort( int a[], int n )` (`n` là số phần tử).

```
Mang goc : 3 5 4 6 7 1 2
Mang giam: 7 6 5 4 3 2 1
```

Bài giải: xem trang 224

Bài 149: Dùng giải thuật đệ quy, viết hàm tìm kiếm nhị phân trên mảng đã được sắp xếp: `int BSearch( int a[], int x, int left, int right )`
(`left` là biên trái, `right` là biên phải của mảng, `x` là số cần tìm trong mảng).

```
2 3 4 5 6 7
Nhap x: 5 ↵
a[3]
```

Bài giải: xem trang 225

[15] Blaise Pascal (1623 - 1662)
[16] Isaac Newton (1643 - 1727)

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 150: Dùng giải thuật đệ quy, viết hàm `int OddSum( int a[], int n )` trả về tổng các phần tử có trị lẻ trong mảng.

```
2 3 4 5 6 7
Tong cac phan tu le: 15
```

Bài giải: xem trang 227

Bài 151: Dùng giải thuật đệ quy, viết hàm `int MaxArr( int a[], int n )` trả về vị trí của phần tử có trị lớn nhất trong một mảng không rỗng.

```
-2 3 -4 -5 6 -8
Max = a[4] = 6
```

Bài giải: xem trang 228

Bài 152: Dùng giải thuật đệ quy, viết hàm `int isSym( int a[], int left, int right )` kiểm tra xem một mảng có đối xứng hay không (`left` là biên trái, `right` là biên phải, trả về `1` nếu mảng đối xứng).

```
7 -2 3 4 3 -2 7
Mang doi xung
```

Bài giải: xem trang 229

Bài 153: Dùng giải thuật đệ quy, viết hàm `float NegAverage( int a[], int n )` trả về trị trung bình các phần tử âm của một mảng chứa các số nguyên.

```
-2 3 -4 -5 6 -8
Trung binh cong cac phan tu am: -4.75
```

Bài giải: xem trang 230

Bài 154: Dùng giải thuật đệ quy, viết chương trình `FACTORIAL.C`, tính giai thừa các số nguyên dương. Các số nguyên dương được nhập vào *từ tham số dòng lệnh*.

```
FACTORIAL 9 3A 12 ↵
9! = 362880
3A: so khong hop le
12! = 479001600
```

Bài giải: xem trang 231

Bài 155: Viết hàm đệ quy `F(n)` tính số Fibonacci thứ `n` (`n` nguyên, `0 < n < 40`, nhập từ bàn phím) theo công thức truy hồi:

$$F_n = \begin{cases} 1 & n = 1,2 \\ F_{n-1} + F_{n-2} & n > 2 \end{cases}$$

Kiểm tra lại bằng công thức dạng đóng (của A. de Moivre) dùng tính số Fibonacci thứ `n` như sau: $F_n = \dfrac{\varphi^n - \psi^n}{\sqrt{5}}$ với $\varphi = \dfrac{1+\sqrt{5}}{2}$, $\psi = 1 - \varphi$ (φ gọi là golden mean).

```
Nhap n (0 < n < 40): 24 ↵
De quy         : Fi(24) = 46368
Cong thuc dong: Fi(24) = 46368
```

Bài giải: xem trang 231

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 156: Viết hàm đệ quy tính phân số liên tục sau:

$$F = 1 + \cfrac{1}{3 + \cfrac{1}{5 + \cfrac{1}{\cdots \; 101 + \cfrac{1}{103}}}}$$

```
F = 1.31304
```

Bài giải: xem trang 232

Bài 157: Viết giải thuật đệ quy, tính định thức của một ma trận vuông bậc `n` (nhập từ bàn phím), trị các phần tử của ma trận lấy ngẫu nhiên trong đoạn `[-10, 10]`.

Công thức truy hồi dùng tính định thức ma trận `A` vuông bậc `n`:

$$\det(A) = \begin{cases} a_{11} & n = 1 \\ \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \det(A_{1j}) & n > 1 \end{cases}$$

Với (A_{ij}) là ma trận con vuông bậc `n-1`, tạo từ ma trận `A` bỏ đi các phần tử trên dòng `1`, cột `j` (dòng và cột của phần tử a_{1j})

```
Nhap bac ma tran: 4 ↵
   10   -2    3    9
    7    6    2  -10
    9   10    4   -5
    3   -4   -3   -2
Det(A) = 2448
```

Bài giải: xem trang 233

Bài 158: Dùng giải thuật đệ quy, viết chương trình `GCD` (Greatest Common Divisor), tính ước số chung lớn nhất của 2 số nguyên thập vào từ bàn phím. Chương trình phải xử lý các trường hợp nhập số `0` hoặc số âm.

Thuật toán Euclid[17] với công thức truy hồi:

$$\gcd(a, b) = \begin{cases} b & a = 0 \\ \gcd(b \bmod a, a) & a > 0 \end{cases}$$

```
Nhap hai so: 0 0 ↵
GCD (0, 0): khong xac dinh
Nhap hai so: 0 3 ↵
GCD (0,3) = 3
Nhap hai so: -18 27 ↵
GCD(-18, 27) = 9
```

Bài giải: xem trang 235

[17] Euclid of Alexandria (325 BC - 265 BC)

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 159: Một palindrome là một từ hay một câu, đọc xuôi ngược đều giống như nhau. Viết chương trình, với giải thuật đệ quy, đọc từng dòng từ bàn phím vào và báo cho biết đó có phải là palindrome không. Với mỗi dòng, bỏ qua các ký tự không phải alphabet và không quan tâm đến chữ hoa, chữ thường.

```
O give me a home ↵
-> khong phai palinedrom
Madam, I'm Adam! ↵
-> Palindrome
^Z ↵
```

Bài giải: xem trang 236

Bài 160: Viết hàm đệ quy in ra tất cả `n!` các hoán vị của chuỗi `n` ký tự đầu bảng alphabet. Áp dụng tìm tất cả các hoán vị của chuỗi `ABC (n = 3)`.

```
Nhap n: 3 ↵
ABC ACB BAC BCA CBA CAB
```

Bài giải: xem trang 236

Bài 161: Viết các hàm đệ quy tính `sin(x)` và `cos(x)` theo cặp đồng nhất thức:

$$\begin{cases} \sin 3x = (4\cos^2 x - 1)\sin x \\ \cos 3x = (1 - 4\sin^2 x)\cos x \end{cases}$$

và cặp đồng nhất thức xấp xỉ cho các giá trị rất nhỏ của `x`:

$$\begin{cases} \sin x \approx x - x^3 / 6 \\ \cos x \approx 1 - x^2 / 2 \end{cases}$$

Với độ chính xác 5.10^{-4}. Kiểm tra với các hàm tương ứng của `math.h`.

```
Doi chung tinh bang math.h trong ()
sin(-2.1) = -0.8632093666 ( -0.8632093666 )
cos(-2.1) = -0.5048461046 ( -0.5048461046 )
```

Bài giải: xem trang 237

Bài 162: In các lần di chuyển đĩa trong bài toán tháp Hanoi với 4 đĩa.

Bài toán tháp Hanoi (Towers of Hanoi): có 3 cọc `A`, `B`, `C`; khởi đầu cọc `A` có `n` đĩa xếp sao cho đĩa lớn hơn luôn nằm bên dưới, 2 cọc kia trống.
Hãy chuyển tất cả đĩa từ cọc `A` sang cọc `C`, được dùng cọc phụ `B`. trong quá trình chuyển đĩa phải đảm bảo đĩa lớn hơn luôn nằm bên dưới.
Minh họa với 2 đĩa:

Xem thêm về bài này trong nhiều sách về lập trình.

```
Disk 1: [1] -> [2]
Disk 2: [1] -> [3]
Disk 1: [2] -> [3]
Disk 3: [1] -> [2]
Disk 1: [3] -> [1]
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
Disk 2: [3] -> [2]
Disk 1: [1] -> [2]
Disk 4: [1] -> [3]
Disk 1: [2] -> [3]
Disk 2: [2] -> [1]
Disk 1: [3] -> [1]
Disk 3: [2] -> [3]
Disk 1: [1] -> [2]
Disk 2: [1] -> [3]
Disk 1: [2] -> [3]
```

Bài giải: xem trang 238

Bài 163: Có thể tính x^n chỉ cần khoảng $\log_2 n$ phép nhân dựa vào nhận xét:

$$x^n = \begin{cases} 1 & n = 0 \\ (x^k)(x^k) & n = 2k \quad k \in N \\ x(x^k)(x^k) & n = 2k + 1 \end{cases}$$

Hãy tính x^n với x là số thực, n là số nguyên, nhập từ bàn phím. Kiểm tra đối chứng bằng hàm `pow()` của `math.h`.

```
Nhap x, n: 8.5 - 2 ↵
mypow()  : 0.013841
pow()    : 0.013841
```

Bài giải: xem trang 239

STRUCTURE - UNION - BIT FIELD

Bài 164: Viết chương trình cài đặt cấu trúc của một số phức và các hàm thực hiện các phép toán trên nó.

Nếu `z = a + bi` và `w = c + di` là hai số phức, thì:

$z + w = (a + c) + (b + d)i$

$z - w = (a - c) + (b - d)i$

$zw = (ac - bd) + (ad + bc)i$

$$\frac{z}{w} = \left(\frac{ac + bd}{c^2 + d^2}\right) + \left(\frac{bc - ad}{c^2 + d^2}\right)i$$

```
Nhap mot so phuc:
  Phan thuc: 3.5 ↵
  Phan ao  : 0.7 ↵
Nhap mot so phuc:
  Phan thuc: 4.2 ↵
  Phan ao  : 2.6 ↵
a + b = 7.7 + 3.3i
a - b = -0.7 - 1.9i
a * b = 12.9 + 12.0i
a / b = 0.7 - 0.3i
```

Bài giải: xem trang 240

Bài 165: Viết chương trình cài đặt cấu trúc của một phân số và các hàm thực hiện các phép toán số học trên nó.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
Nhap tu so va mau so: 1 -2 ↵
Nhap tu so va mau so: -3 4 ↵
a + b = -5/4
a - b = 1/4
a * b = 3/8
a / b = 2/3
```

Bài giải: xem trang 242

Bài 166: Viết chương trình cài đặt cấu trúc lưu thông tin của đường tròn, bao gồm tọa độ tâm và bán kính của đường tròn. Nhập dữ liệu của hai đường tròn, xác định vị trí tương đối giữa chúng.

Với mỗi cặp đường tròn, xét 3 trường hợp:
- Tâm đường tròn này nằm ngoài đường tròn kia.
- Hai tâm trùng nhau.
- Tâm đường tròn này nằm trên hoặc trong đường tròn kia (trừ tâm).

Dựa vào khoảng cách giữa hai tâm với bán kính của mỗi đường tròn để xác định vị trí tương đối giữa chúng.

```
Nhap xc, yc va R cua C1: 2.23 1.12 1.67 ↵
Nhap xc, yc va R cua C2: 1.32 2.41 3.25 ↵
Trong nhau
```

Bài giải: xem trang 243

Bài 167: Viết chương trình cài đặt cấu trúc một đơn thức (một tham số), và dùng mảng các cấu trúc này để lưu một đa thức. Nhập vào hai đa thức, thực hiện phép toán nhân hai đa thức này; sau đó tính trị của đa thức kết quả với trị của tham số nhập vào từ bàn phím.

Tích của 2 đa thức 1 tham số x, f(x) bậc m và g(x) bậc n là:

$f(x).g(x) = c_{m+n}x^{m+n} + c_{m+n-1}x^{m+n-1} + ... + c_1x + c_0$

Trong đó c_k bằng tổng các tích a_ib_j mà $i + j = k$ $(k = 0,1,...,m + n)$

Tính trị đa thức tổng quát bậc n: $P(x) = a_nx^n + a_{n-1}x^{n-1} + ... + a_1x + a_0$ phải thực hiện n(n+1)/2 phép nhân (x^k tính k-1 phép nhân). Nếu tính trị của đa thức bằng phương pháp Horner sau đây, chỉ cần thực hiện n phép nhân:

$P(x) = ((...((a_nx + a_{n-1})x + a_{n-2})x + ... + a_2)x + a_1)x + a_0$

```
Nhap bac da thuc: 3 ↵
Nhap 4 he so: 5 -4 7 -2
Nhap bac da thuc: 2 ↵
Nhap 3 he so: 3 -2 7 ↵
p1(x) = 5x^3-4x^2+7x-2
p2(x) = 3x^2-2x+7
Da thuc ket qua:
p(x) = 15x^5-22x^4+64x^3-48x^2+53x-14
Nhap x: 1.2 ↵
p(1.2) = 82.7776
```

Bài giải: xem trang 244

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 168: Viết chương trình cài đặt một mảng các cấu trúc lưu trữ thông tin sách trong thư viện, bao gồm: tựa sách, ISBN, tên tác giả, tên nhà xuất bản, ngày tháng năm nhập sách (là ngày viết phiếu). Sau đó, nhập vào một chuỗi ISBN, tìm và in ra thông tin sách tương ứng nếu có.

Dùng `struct tm` của `time.h` để lưu trữ ngày cập nhật phiếu.

```
Nhap thong tin sach:
 Tua     > The C Programming Language ↵
 ISBN    > 0-131-10370-9 ↵
 Tac gia > Kernighan, Brian W. & Ritchie, Dennis M. ↵
 NXB     > Prentice Hall ↵
Tiep ( y/n )? y ↵
Nhap thong tin sach:
 Tua     > Applications Programming in ANSI C ↵
 ISBN    > 0-023-61141-3
 Tac gia > Johnsonbaugh, R. & Kalin M. ↵
 NXB     > MacMillan Pub. Co. ↵
Tiep ( y/n )? n ↵
ISBN ? 0-023-61141-3 ↵
Ket qua tim:
Applications Programming in ANSI C
 Johnsonbaugh, R. & Kalin M.
 MacMillan Pub. Co.
 [update: 15-04-2006]
```

Bài giải: xem trang 246

Bài 169: Viết chương trình cài đặt một cấu trúc lưu thông tin chấm công ngày (time card) của một tổ 6 công nhân, bao gồm: mã số công nhân, giờ vào làm và giờ nghỉ. Một đợt vào làm không được làm vượt quá 2 ca liên tiếp. In bảng thanh toán công ngày của tổ công nhân đó. Biết:

Ca đêm	01 giờ - 06 giờ, 18 giờ - 24 giờ	15000đ/giờ
Ca ngày	06 giờ - 18 giờ	10000đ/giờ

```
Nhap ID cong nhan 1: 78-125 ↵
Vao (gio phut): 1 45 ↵
Ra (gio phut) : 5 30 ↵
...
Nhap ID cong nhan 6: 89-273 ↵
Vao (gio phut): 21 25 ↵
Ra (gio phut) : 3 40 ↵
  78-125  01:45  05:30   56250
  45-224  03:10  09:50   80833
  52-436  06:20  01:30   Nhap sai!
  82-729  14:40  20:30   70833
  67-451  18:45  23:10   70000
```

Bài giải: xem trang 248

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 170: Viết chương trình đặt một cấu trúc lưu 52 lá bài của bộ bài tây, bao gồm nước (face: ách , hai, ba, …, mười, bồi, đầm, già) và chất (suit: cơ, rô, chuồn, bích). Thực hiện việc chia bài ngẫu nhiên cho 4 người chơi.

Hãy trộn bài (shuffle) *ngẫu nhiên*, trước khi chia theo *thứ tự*.

```
 Chin Ro      Bon Co      Bay Bich    Dam Ro
  Sau Co      Bay Ro      Nam Ro       Ba Ro
  ...
  Gia Chuon   Boi Ro      Nam Chuon  Chin Bich
  Bon Chuon   Sau Ro      Ach Ro      Sau Bich
```

Bài giải: xem trang 250

Bài 171: Viết chương trình nhập vào một ký tự. Không dùng các toán tử thao tác trên bit (bitwise operators) cũng như không thực hiện phép chia cho 2, hãy in dạng hiển thị nhị phân của ký tự này.

Dùng `union` kết hợp với `bit field`.

```
Nhap 1 ky tu: Z ↵
0 1 0 1 1 0 1 0
```

Bài giải: xem trang 251

TẬP TIN

Bài 172: Viết chương trình `FILESIZE.C` dùng xác định kích thước của một số tập tin. Tên các tập tin cần được xác định kích thước được nhập như là tham số dòng lệnh của `FILESIZE`.

```
FILESIZE readme.txt list.dat ↵
readme.txt [2148 byte(s)]
list.dat [1246 byte(s)]
```

Bài giải: xem trang 252

Bài 173: Viết chương trình ghi 5 số nguyên ngẫu nhiên vào tập tin `INTERGER.DAT`, 5 số thực ngẫu nhiên vào tập tin `REAL.DAT`. Sau đó đọc các số này từ tập tin và xuất ra màn hình.

```
Ghi xong file...
   28764     13997     19450      7297     24082
 0.14554   0.13376   0.82284   0.35276   0.79724
Doc xong file...
```

Bài giải: xem trang 253

Bài 174: Tập tin văn bản `PERSON.DAT` lưu thông tin cá nhân thành các dòng có định dạng như sau (chú thích trong () là yêu cầu của trường):

`code`(unsigned int)`:firstname lastname`(32)`,address`(32)`:birthday`(mm/dd/yy)

Viết chương trình đọc tập tin `PERSON.DAT`, lấy và hiển thị thông tin lưu trữ ứng với từng cá nhân.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
Nội dung tập tin PERSON.DAT
1654:Jackie Chan,Hong Kong:7/22/54
4424:Tony Jaa,Thailand:5/12/76
Kết quả
Jackie Chan                    [code: 1654]
  Address : [Hong Kong]
  Birthday: [7/22/54]
Tony Jaa                       [code: 4424]
  Address : [Thailand]
  Birthday: [5/12/76]
```

Bài giải: xem trang 254

Bài 175: Viết chương trình `UPPER.C` khi chạy sẽ chuyển đổi tất cả ký tự thường của một tập tin (nhập tên tập tin từ dòng lệnh) thành ký tự hoa chứa vào một tập tin khác (nhập tên tập tin từ dòng lệnh).

```
UPPER lower.txt upper.txt ↵
Chuyen thanh chu hoa xong...
```

Bài giải: xem trang 256

Bài 176: Viết chương trình `CIPHER.C` có hai chức năng: mã hóa nội dung một tập tin và giải mã nội dung tập tin được mã hóa. Luật mã hóa: ký tự nào có mã ASCII nhỏ hơn 128 thì chuyển mã thành: mã ký tự đó + 128; ký tự nào có mã ASCII lớn hơn 128 thì chuyển mã thành: mã ký tự đó - 128.

```
Mã hóa:
CIPHER origin.txt encode.txt ↵
Xu ly xong...
Giải mã:
CIPHER encode.txt origin.txt ↵
Xu ly xong...
```

Bài giải: xem trang 257

Bài 177: Viết chương trình `TESTFILE.C` sử dụng theo cú pháp sau:

`TESTFILE num filename` : (*number*) tạo tập tin `filename` chứa `num` ký tự ngẫu nhiên viết hoa.

`TESTFILE -v filename` : (*view*) in nội dung tập tin `filename` ra màn hình.

`TESTFILE -r filename` : (*reverse*) in ngược nội dung tập tin `filename` ra màn hình.

```
TESTFILE 10 test.txt ↵
File da duoc tao...
TESTFILE -v test.txt ↵
F L P C Q P Z D L O
Cuoi file...
TESTFILE -r test.txt ↵
O L D Z P Q C P L F
Dau file...
```

Bài giải: xem trang 258

Bài 178: Viết chương trình `LINES.C` hiển thị nội dung của tập tin nguồn với từng dòng được đánh số tại đầu dòng. Cú pháp sử dụng:

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

`LINES < SOURCE.C` : hiển thị tập tin `SOURCE.C` có đánh số dòng.
`LINES < SOURCE.C > NSOURCE.C`: tập tin mới `NSOURCE.C` sẽ có đánh số dòng.

```
LINES < SOURCE.C ↵
  0: #include <stdio.h>
  1:
  2: int main ()
  3: {
     ...
 17:   return 0;
 18: }
```

Bài giải: xem trang 259

Bài 179: Viết chương trình thực hiện các tác vụ sau:
- Tạo tập tin chứa các mẫu tin theo cấu trúc sau:

```
struct STUDENT {
  int code;                    /* mã số */
  char name [20];              /* tên */
  double avgmark;              /* điểm trung bình */
};
```

- Nhập thông tin một số sinh viên, thêm vào tập tin (giả sử nhập trường `code` không trùng, không cần kiểm tra).
- In tất cả các mẫu tin lưu trong tập tin.
- Nhập `code`, tìm mẫu tin trong tập tin theo `code`; nếu tìm được, cho phép cập nhật lại trường `avgmark` của mẫu tin đó.
- Dùng một menu đơn giản để hiển thị các tác vụ.

```
Ten file? STUDENT.DAT ↵

MENU (File 'STUDENT.DAT')
----
[1]. Them
[2]. Doc
[3]. Sua
[4]. Thoat
Chon tac vu: 1 ↵
GHI FILE
Nhap mot mau tin (y/n)? y ↵
 Ma so   > 7366 ↵
 Ten     > Steve Blenheim ↵
 Diem TB > 7.84 ↵
Nhap mot mau tin (y/n)? y ↵
 Ma so   > 8376 ↵
 Ten     > James Ikeda ↵
 Diem TB > 8.21 ↵
Nhap mot mau tin (y/n)? n ↵
Da ghi file...

MENU (File 'STUDENT.DAT')
----
[1]. Them
[2]. Doc
[3]. Sua
[4]. Thoat
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
Chon tac vu: 2 ↵
DOC FILE
1  7366 Steve Blenheim  7.84
2  8376 James Ikeda     8.21
Tong cong: 2 record(s)

MENU (File 'STUDENT.DAT')
----
[1]. Them
[2]. Doc
[3]. Sua
[4]. Thoat
Chon tac vu: 3 ↵
SUA
Ma so > 8376 ↵
  James Ikeda
Diem moi > 7.94 ↵
Da cap nhat...

MENU (File 'STUDENT.DAT')
----
[1]. Them
[2]. Doc
[3]. Sua
[4]. Thoat
Chon tac vu: 4 ↵
Bye...
```

Bài giải: xem trang 260

Bài 180: Viết chương trình đọc một tập tin văn bản. Sau đó trình bày những thống kê sau: số dòng, số từ, tần số xuất hiện của các từ trong tập tin đó.

Nội dung tập tin HANOI.TXT

```
Ta con em, cay bang mo coi mua Dong...
Goc pho mo coi mua Dong, manh trang mo coi mua Dong
Mua Dong nam ay...
```

Kết quả:

```
Nhap ten file: HANOI.TXT ↵
File HANOI.TXT co 3 dong, 25 tu, voi tan so cac tu:
      ta:  1      con:  1       em:  1      cay:  1
    bang:  1       mo:  3      coi:  3      mua:  4
    dong:  4      goc:  1      pho:  1     manh:  1
   trang:  1      nam:  1       ay:  1
```

Bài giải: xem trang 263

Bài 181: Viết chương trình nhận vào chuỗi s, sau đó đọc một tập tin văn bản. In số thứ tự các dòng có chứa chuỗi s; với mỗi dòng đó in các vị trí có xuất hiện chuỗi s.

Nội dung tập tin TONGBIET.TXT

```
Dua nguoi, ta khong dua sang song,
Sao lai thay song o trong long?
Bong chieu khong tham, khong vang vot
Sao day hoang hon trong mat trong?
Kết quả:
Nhap ten file: THO.TXT ↵
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
Nhap chuoi tim: khong ↵
Dong 0: 14
Dong 2: 11 23
```

Bài giải: xem trang 265

Bài 182: Viết chương trình tạo một tập tin nhị phân chứa các mẫu tin, mỗi mẫu tin bao gồm tên, số điện thoại, địa chỉ của một người. Sau đó đọc tập tin nhị phân này và tạo một tập tin văn bản chứa kết quả đọc được, trình bày ở dạng bảng.

```
Tao file nhi phan PERSON.DAT...
Nhap mot mau tin (y/n)? y ↵
Ten        > Popeye Sailor ↵
Dien thoai > (156)454-3325 ↵
Dia chi    > 94 Comics Str., Anywhere ↵
Nhap mot mau tin (y/n)? y ↵
Ten        > James Bond ↵
Dien thoai > (846)233-2837 ↵
Dia chi    > 07 Movies Str., Cinema ↵
Nhap mot mau tin (y/n)? n ↵
Tao file van ban PERSON.TXT...
Hien thi file van ban PERSON.TXT...
Popeye Sailor  (156)454-3325  94 Comics Str., Anywhere
James Bond     (846)233-2837  07 Movies Str., Cinema
```

Bài giải: xem trang 265

Bài 183: Viết một hàm dùng giải thuật đệ quy đọc các dòng văn bản (không quá 255 ký tự) từ một tập tin. Sau đó ghi các dòng văn bản này theo một thứ tự đảo ngược vào một tập tin khác.

```
REVERSE test.txt reverse.txt ↵
Dao nguoc de quy xong...
```

Bài giải: xem trang 267

Bài 184: Tập tin `EMP.DAT` chứa các hồ sơ nhân viên, mỗi hồ sơ là một cấu trúc bao gồm tên, ngày sinh (là một cấu trúc bao gồm ngày, tháng, năm sinh), lương. Viết chương trình nhập, sắp xếp các mẫu tin trong tập tin trên theo thứ tự tăng của tuổi, xuất danh sách đã sắp xếp. Thao tác trên tập tin, không dùng mảng tạm.

```
Nhap so nhan vien: 3 ↵
Nhap (ten, ngay, thang, nam sinh, luong):
1 > Dong 30 4 1982 350000 ↵
2 > Tay 26 3 1980 150000 ↵
3 > Nam 1 9 1982 400000 ↵
Nhap du lieu xong ...
Xuat danh sach sap xep:
Nam      1/9/1982        400000
Dong     30/4/1982       350000
Tay      26/3/1980       150000
```

Bài giải: xem trang 268

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 185: Mỗi ma trận kích thước `m x n` được lưu trong một tập tin văn bản theo quy ước sau: dòng đầu chứa trị `m` (số dòng) và trị `n` (số cột); `m` dòng tiếp theo, mỗi dòng lưu `n` phần tử trên các dòng tương ứng của ma trận, cách nhau bởi dấu space.
Viết chương trình nhận các tham số dòng lệnh là tên các tập tin văn bản, thực hiện công việc sau:
- Nếu có hai tham số dòng lệnh: tạo hai ma trận (kích thước nhập từ bàn phím), có các phần tử là các số thực ngẫu nhiên và lưu vào hai tập tin văn bản có tên tương ứng với các tham số dòng lệnh.
- Nếu có ba tham số dòng lệnh: nhân hai ma trận chứa trong các tập tin văn bản có tên là tham số dòng lệnh thứ nhất và thứ hai, sau đó lưu ma trận kết quả vào tập tin văn bản có tên là tham số dòng lệnh thứ ba. Nếu phép nhân ma trận không thực hiện được, không tạo tập tin kết quả.

```
MULMATRIX MATRIX1.TXT MATRIX2.TXT ↵
Nhap m, n: 3 4 ↵
Tao xong file chua ma tran...
Nhap m, n: 4 5 ↵
Tao xong file chua ma tran...

MULMATRIX MATRIX1.TXT MATRIX2.TXT MATRIX3.TXT ↵
Tao xong file chua ma tran...
```

Bài giải: xem trang 270

Bài 186: Thông tin một quyển sách trong thư viện được lưu trong tập tin văn bản `BOOKS.TXT` thành một khối như sau:

```
C How to Program
Deitel, H.M. & Deitel, P.J.
Prentice Hall, 2001
ISBN 0-13-089572-5
*
```

Các khối thông tin lưu liên tục nhau và cách nhau bởi dòng có dấu `*` (các dòng khác trong khối không được có dấu `*` đầu dòng)
Để tìm kiếm nhanh, người ta tạo một tập tin nhị phân `INDEX.IDX` lưu các mẫu tin gồm tên sách và vị trí của khối thông tin về tên sách đó trong tập tin `BOOKS.TXT`.
Viết chương trình `LOOK` sử dụng theo cú pháp sau:

`LOOK -i BOOKS.TXT INDEX.IDX` : Tạo tập tin `INDEX.IDX`.
`LOOK -v BOOKS.TXT INDEX.IDX` : Hỏi tên sách, tìm trong `BOOKS.TXT` nhanh nhờ `INDEX.IDX`, in thông tin.

```
LOOK -i BOOKS.TXT INDEX.IDX ↵
Tao xong file index...
LOOK -v BOOKS.TXT INDEX.IDX ↵
Nhan Ctrl+Z de dung
Ten sach? C How to Program ↵
  C How to Program
  Deitel, H.M & Deitel, P.J.
  Prentice Hall, 2001
  ISBN 0-13-089572-5
Ten sach? ^Z ↵
```

Bài giải: xem trang 273

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 187: Viết chương trình hiển thị nội dung một tập tin văn bản trên thiết bị xuất chuẩn (giả sử là màn hình văn bản 80 x 25); mỗi lần hiển thị đầy các dòng của màn hình, sau đó dừng lại chờ nhấn phím Enter để hiển thị tiếp.
Bài giải: xem trang 275

Bài 188: Cho hai tập tin văn bản chứa các số nguyên được sắp xếp theo thứ tự tăng dần. Viết một chương trình để trộn (merge) hai tập tin này vào một tập tin số nguyên thứ ba cũng được sắp theo thứ tự tăng dần (không dùng mảng phụ). Tên của cả ba tập tin được nhập vào như các tham số dòng lệnh.

```
Nội dung tập tin NUMBER1.TXT: 1 5 9 13 17 19
Nội dung tập tin NUMBER2.TXT: 2 6 10 14 18 22
SORT NUMBER1.TXT NUMBER2.TXT NUMBER.TXT ↵
Tron ket thuc...
Nội dung tập tin NUMBER.TXT: 1 2 5 6 9 10 13 14 17 18 19 22
```

Bài giải: xem trang 276

Bài 189: Viết chương trình chia (split) một tập tin thành nhiều tập tin nhỏ có kích thước tối đa `n`. Các tập tin nhỏ này cùng tên với tập tin ban đầu, với phần mở rộng được đánh số.
Chương trình nhận hai tham số dòng lệnh: tên tập tin cần chia và kích thước tối đa `n` (tính bằng `Kb`).

Kiểm tra kết quả bằng cách nối các tập tin lại (lệnh `COPY /B`) hoặc viết chương trình kết hợp (combine) các tập tin con đã chia ở trên thành một tập tin duy nhất.

```
Tập tin bigfile.dat có kích thước 58436 bytes
SPLIT bigfile.dat 20 ↵
File bigfile.000 [20480 byte(s)]
File bigfile.001 [20480 byte(s)]
File bigfile.002 [17476 byte(s)]
Chia file ket thuc... [3 file(s)]
```

Bài giải: xem trang 277

Bài 190: Hiển thị nội dung của một tập tin dưới dạng số thập lục phân. Cách trình bày này (gọi là "hex dump") thường thấy trong các tiện ích xem tập tin phổ biến. Xem ví dụ minh họa.

```
         +0 +1 +2 +3 ... +C +D +E +F  Contents
00000000 23 69 6E 63 ... 64 69 6F 2E  #include <stdio.
00000010 68 3E 0D 0A ... 20 3C 63 74  h>  #include <ct
00000020 79 70 65 2E ... 69 7A 65 5F  ype.h>     size_
...
00000450 7D 20 0D 0A                  }
1108 bytes
```

Bài giải: xem trang 278

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

CÁC VẤN ĐỀ KHÁC

Bài 191: Dùng các hàm chuẩn của `time.h` viết hàm nhập vào tháng và năm (sau 1900). In lịch bằng tiếng Việt (không dấu) của tháng đó chỉ trong 5 dòng.

```
Nhap thang, nam (sau 1900): 7 2006 ↵
Thang Bay 2006
   CN  Hai   Ba   Tu  Nam  Sau  Bay
   30   31                       1
    2    3    4    5    6    7    8
    9   10   11   12   13   14   15
   16   17   18   19   20   21   22
   23   24   25   26   27   28   29
```

Bài giải: xem trang 279

Bài 192: Dùng các hàm chuẩn của `time.h`, viết chương trình hiển thị thứ, ngày, tháng, năm, giờ hiện tại, múi giờ GMT, bằng tiếng Việt (không dấu).

```
Hom nay Chu Nhat, ngay 09 thang Hai nam 2003
Bay gio la 05:24:18 AM +007 GMT
```

Bài giải: xem trang 281

Bài 193: Tạo menu cho chương trình trong bài 178 (trang 50) bằng cách dùng mảng các con trỏ hàm (functor).

```
MENU (File 'STUDENT.DAT')
----
[1]. Them
[2]. Doc
[3]. Sua
[4]. Thoat
Chon tac vu:
```

Bài giải: xem trang 283

Bài 194: Viết một hàm nhận danh sách tham số gồm: tên một hàm lượng giác (có một tham số `double`) của `math.h`, cận dưới và cận trên của miền cần tính trị, bước tăng khi tính trị. In ra bảng các trị của hàm đó trong miền và bước tăng được chọn.

```
Gọi hàm khi cần tính sin(x) trong đoạn [0, π], với bước tăng π/8:
tab_func( sin, 0.0, M_PI, M_PI/8.0 );
Kết quả:
         x       f(x)
    0.0000     0.0000
    0.3927     0.3827
    0.7854     0.7071
    1.1781     0.9239
    1.5708     1.0000
    1.9635     0.9239
    2.3562     0.7071
    2.7489     0.3827
    3.1416     0.0000
```

Bài giải: xem trang 285

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 195: Viết một hàm `Average( n, ... )`, nhận vào `n` tham số nguyên (sau tham số thứ nhất là `n`), trả về trung bình cộng của `n` tham số nguyên đó.

```
x = 32.4   y = 24.7   z = 4.5   t = 11.8
average( 2, x, y )      : 28.550
average( 3, x, y, z )   : 20.533
average( 4, x, y, z, t ): 18.350
```

Bài giải: xem trang 285

Bài 196: Viết các macro:

`islower( c )`	cho kết quả là `1` nếu ký tự `c` là chữ thường, hoặc `0` nếu ký tự `c` là chữ hoa.
`toupper( c )`	cho kết quả là ký tự hoa tương ứng với `c` nếu `c` là chữ thường, hoặc cho kết quả là chính ký tự `c` nếu `c` là chữ hoa.
`percent( a, b )`	cho kết quả là phần trăm của `a` trên `b`.

```
'c' viet hoa la 'C'
123 la 0.996355% cua 12345
```

Bài giải: xem trang 286

Bài 197: Viết các macro:

`bitOff( d, n )`	đặt bit thứ `n` của `d` về `0`.
`bitOn( d, n )`	bật bit thứ `n` của `d` lên `1`.
`bitFlip( d, n )`	đảo bit thứ `n` của `d`.
`isBit( d, n )`	trả về `1` nếu bit thứ `n` của `d` đang là `1` hoặc trả về `0` nếu bit thứ `n` này đang là `0`.

Với `d` là số nguyên.

Tạo rồi dùng macro `isBit( d, n )` để tạo hàm tạo in các bit của một số nguyên.

```
So goc   : 12345 = 00000000 00000000 00110000 00111001
Bat bit 8: 12601 = 00000000 00000000 00110001 00111001
Xoa bit 5: 12313 = 00000000 00000000 00110000 00011001
Dao bit 4: 12329 = 00000000 00000000 00110000 00101001
```

Bài giải: xem trang 286

Bài 198: Viết macro `hibyte( n )` và `lowbyte( n )` trả về byte cao và byte thấp của số integer `n`.

```
Platform 32-bits
INT_MAX / 10:
214748364 = 00001100 11001100 11001100 11001100
High Byte: 3276 = 00000000 00000000 00001100 11001100
Low Byte : 52428 = 00000000 00000000 11001100 11001100
```

Bài giải: xem trang 288

Bài 199: Tập tin văn bản `EMP.TXT` lưu thông tin nhân viên thành các dòng với định dạng sau:

```
ID:firstname lastname:birthday:salary
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Các trường cách nhau bởi dấu ":" ; `firstname` và `lastname` cách nhau bởi dấu `space`; cuối dòng có ký tự `newline`.
Viết chương trình đọc tập tin `EMP.TXT`, hiển thị thông tin các nhân viên với `firstname` sắp xếp theo thứ tự alphabet, nếu `firstname` giống nhau, sắp xếp theo `lastname`. Thao tác sắp xếp phải sử dụng hàm chuẩn `qsort()` của `stdlib.h`.

```
Noi dung...
4424:Tom Jones:5/12/66:54335
2638:Jackie Lee:9/23/44:42500
1683:Billy Black:9/23/44:336500
1654:Jackie Chan:7/22/54:65000
5346:Mary Adams:11/4/63:28765
Sap xep...
1683:Billy Black:9/23/44:336500
1654:Jackie Chan:7/22/54:65000
2638:Jackie Lee:9/23/44:42500
5346:Mary Adams:11/4/63:28765
4424:Tom Jones:5/12/66:54335
```

Bài giải: xem trang 288

Bài 200: Tập tin văn bản `PERSON.TXT` lưu thông tin cá nhân thành các dòng với định dạng sau:

`ID:firstname lastname:(area_code) phone_number`

Các trường cách nhau bởi dấu ":", các cặp `firstname` và `lastname`, `area_code` (bao quanh bởi cặp ngoặc đơn) và `phone_number`, đều cách nhau bởi dấu `space`, cuối dòng có ký tự `newline`.
Viết chương trình đọc tập tin `PERSON.TXT`, nhập vào `firstname` của một người (giả sử các `firstname` đều khác nhau), tìm và hiển thị mã vùng của người đó. Thao tác tìm kiếm phải sử dụng hàm chuẩn `bsearch()` của `stdlib.h`.

```
Noi dung...
4424:Mike Harrington:(510) 5481278
2638:Christian Dobbins:(408) 5382358
5346:Jody Savage:(311) 5481278

Nhap firstname: Christian
Ma vung cua Christian: 408
```

Bài giải: xem trang 291

CẤU TRÚC DỮ LIỆU

Các bài tập cấu trúc dữ liệu (data structure) được đưa vào tập sách này được xem như là các bài tập dùng rèn luyện: thao tác trên con trỏ, truyền tham số bằng con trỏ (hiệu quả tương đương với truyền tham số bằng tham chiếu trong C++), giải thuật đệ quy…
Vì lý do trên, phần này chỉ giới hạn trong phạm vi các bài tập với danh sách liên kết đơn và cây tìm kiếm nhị phân (BST).
Theo quan điểm riêng của chúng tôi, bài tập cấu trúc dữ liệu được thực hiện bằng C++ hoặc Java với sự hỗ trợ của OOP sẽ dễ dàng và thuận lợi hơn. Tuy nhiên việc viết lại các bài tập trong phần này bằng C++ hoặc Java không khó.
Phần lớn các bài tập trong phần danh sách liên kết đơn đều dùng thông tin lưu tại

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

mỗi node (trừ các liên kết) là một số nguyên.

```
struct NODE {
  int data;
  struct NODE* next;
};
typedef struct NODE* NODEPTR;
```

Thông tin lưu trữ phức tạp hơn chỉ ảnh hưởng đến việc truy xuất, không ảnh hưởng đến giải thuật. Ví dụ:

```
struct STUDENT {
  char name[30];
  int age;
};
typedef struct STUDENT* SLINK;

struct NODE {
  SLINK data;
  struct NODE* next;
};
typedef struct NODE* NODEPTR;
```

Nên tổ chức và quản lý thông tin lưu trữ như trên để thuận tiện cho việc hoán chuyển dữ liệu khi cần.

Phần lớn các bài tập trong phần BST đều dùng cấu trúc dữ liệu chung như sau:

```
struct NODE {
  int data;
  struct NODE *left, *right;
};
typedef struct NODE* NODEPTR;
```

Chúng tôi cũng dùng một cây có cấu trúc tương đối tổng quát để dễ kiểm tra kết quả xuất của các bài tập trong phần này, dữ liệu nhập: `9 3 16 14 2 5 4 8 18`

Vì các bài giải dài và có nhiều tác vụ lặp đi lặp lại nên chỉ có một số bài giải đầy đủ, các bài giải còn lại chỉ tập trung giải quyết yêu cầu. Bài giải đầy đủ xin tham khảo trong source code của sách.

Bài 201: Xét các trường hợp chèn một node mới vào một danh sách liên kết đơn chứa các trị nguyên.

```
Nhap 0 de dung: 3 2 1 0 ↵
List goc: [1][2][3]
Chen 5 cuoi: [1][2][3][5]
Chen 4 sau node [3]: [1][2][3][4][5]
```

Bài giải: xem trang 296

Bài 202: Cho một danh sách liên kết đơn chứa các trị nguyên. Dùng giải thuật đệ quy, chèn một node chứa trị `0` sau mỗi node chứa trị chẵn trong danh sách liên kết.

```
Nhap 0 de dung: 1 2 4 3 5 6 0 ↵
List goc: [1][2][4][3][5][6]
Chen 0 sau tri chan: [1][2][0][4][0][3][5][6][0]
```

Bài giải: xem trang 300

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 203: Chèn một node mới vào một danh sách liên kết đơn, sao cho trị lưu trong các node của danh sách luôn theo thứ tự tăng. Không dùng đệ quy.

```
Nhap 0 de dung: 3 2 5 1 0 ↵
List goc: [1][2][3][5]
Nhap tri moi: 4 ↵
List moi: [1][2][3][4][5]
```

Bài giải: xem trang 301

Bài 204: Xóa một node trong một danh sách liên kết đơn chứa các trị nguyên. Xét các trường hợp phổ biến.

```
Nhap 0 de dung: 1 2 3 4 0 ↵
List goc: [1][2][3][4]
Nhap tri can xoa: 3 ↵
List moi: [1][2][4]
```

Bài giải: xem trang 302

Bài 205: Cho một danh sách liên kết đơn chứa các trị nguyên. Viết hàm thực hiện xóa lần lượt tất cả các node từ cuối danh sách ngược trở lên đầu.

```
Nhap 0 de dung: 1 2 3 0 ↵
List goc: [1][2][3]
Xoa node [3]
Xoa node [2]
Xoa node [1]
```

Bài giải: xem trang 304

Bài 206: Cho một danh sách liên kết đơn chứa các trị nguyên. Viết hàm thực hiện xóa tất cả các node có trị chỉ định nhập từ bàn phím.

```
Nhap 0 de dung: 1 2 1 1 3 1 4 1 1 0 ↵
List goc: [1][2][1][1][3][1][4][1][1]
Nhap tri can xoa: 1 ↵
List moi: [2][3][4]
```

Bài giải: xem trang 304

Bài 207: Cho một danh sách liên kết đơn chứa các trị nguyên, sửa các liên kết của danh sách này để có được hai danh sách danh sách liên kết mới: một danh sách chứa các trị lẻ của danh sách đã cho, một danh sách chứa trị chẵn của danh sách đã cho.

```
Nhap 0 de dung: 1 2 3 4 5 6 7 8 0 ↵
List goc : [1][2][3][4][5][6][7][8]
List chan: [8][6][4][2]
List le  : [7][5][3][1]
```

Bài giải: xem trang 305

Bài 208: Cho một danh sách liên kết đơn chứa các trị nguyên. Đổi `m` phần tử cuối của danh sách liên kết lên đầu, `m` là số nguyên dương cho trước.

```
Nhap 0 de dung: 1 2 3 4 5 0 ↵
List goc: [1][2][3][4][5]
Nhap m: 2 ↵
List moi: [4][5][1][2][3]
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài giải: xem trang 306

Bài 209: Cho một danh sách liên kết đơn chứa các trị nguyên (không chứa nhiều phần tử). In các phần tử của danh sách theo thứ tự ngược lại với thứ tự lưu trữ kể từ đầu danh sách. Tìm phần tử chứa trị lớn nhất trong danh sách bằng đệ quy.

```
Nhap 0 de dung: 1 3 5 2 4 0 ↵
List dao: [4][2][5][3][1]
Tri max: 5
```

Bài giải: xem trang 307

Bài 210: Đảo ngược các liên kết của một danh sách liên kết đơn chứa các trị nguyên.

```
Nhap 0 de dung: 1 2 3 4 5 0 ↵
List goc: [1][2][3][4][5]
List dao: [5][4][3][2][1]
```

Bài giải: xem trang 308

Bài 211: Cho danh sách liên kết đơn chứa các trị nguyên. Tách danh sách này thành các danh sách liên kết đơn con, mỗi danh sách liên kết đơn con chứa một "run" tăng. Dùng một danh sách liên kết đơn riêng quản lý các danh sách "run" nói trên.

```
Nhap 0 de dung: 1 2 3 2 3 4 5 4 6 0 ↵
List 'run':
r-+-[4][6][n]
  |-[2][3][4][5]
  `-[1][2][3][n]
```

Bài giải: xem trang 309

Bài 212: Cho danh sách liên kết đơn quản lý các "run" như bài 211 (trang 61). Trộn các "run" do danh sách này quản lý thành một "run" tăng duy nhất.

```
Nhap 0 de dung: 1 2 3 2 3 4 5 4 6 0 ↵
List 'run':
r-+-[4][6][n]
  |-[2][3][4][5][n]
  `-[1][2][3][n]
'run' tang duy nhat:
r-+-[n]
  |-[n]
  `-[1][2][2][3][3][4][4][5][6][n]
```

Bài giải: xem trang 311

Bài 213: Tìm phần tử chứa trị xuất hiện nhiều lần nhất (tần suất cao nhất) trong một danh sách liên kết đơn khác rỗng chứa các trị nguyên, giả sử tần suất của các trị trong danh sách liên kết đều khác nhau.

```
Nhap 0 de dung: 1 2 3 3 4 3 2 2 4 2 0 ↵
List goc: [1][2][3][3][4][3][2][2][4][1]
Phan tu xuat hien nhieu nhat: [2](4)
```

Bài giải: xem trang 312

Bài 214: Cho một danh sách liên kết đơn chứa các trị nguyên. Thay đổi các liên kết của danh sách sao cho: các node chứa trị chẵn nằm đầu danh sách, các node chứa trị

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

lẻ nằm cuối danh sách, thứ tự xuất hiện các node giống với danh sách ban đầu.

```
Nhap 0 de dung: 3 8 4 1 5 7 6 2 0 ↵
List goc: [3][8][4][1][5][7][6][2]
List moi: [8][4][6][2][3][1][5][7]
```

Bài giải: xem trang 313

Bài 215: Cho một danh sách liên kết đơn không rỗng, viết chương trình thay đổi các liên kết của danh sách sao cho danh sách trở thành:

```
node[1] ⇒ node[3] ⇒ ... ⇒ node[2n+1]
                        ⇒ node[2] ⇒ node[4] ⇒ ... ⇒ node[2n] ⇒ NULL
```

```
Nhap 0 de dung: 4 3 7 2 6 5 9 8 0 ↵
List goc: [4][3][7][2][6][5][9][8]
List moi: [4][7][6][9][3][2][5][8]
```

Bài giải: xem trang 314

Bài 216: Thực hiện giải thuật Selection Sort trên danh sách liên kết đơn chứa các trị nguyên, sắp xếp tăng các phần tử chứa trong danh sách.

```
Nhap 0 de dung: 1 3 5 7 2 4 6 8 0 ↵
List goc : [1][3][5][7][2][4][6][8]
List tang: [1][2][3][4][5][6][7][8]
```

Bài giải: xem trang 315

Bài 217: Cho một danh sách liên kết đơn chứa các trị nguyên. Đảo các node trong danh sách liên kết chứa trị `k` với node ngay sau nó. Trong đó, *hoán chuyển các node* thay vì hoán chuyển dữ liệu trong node.

```
Nhap 0 de dung: 1 2 3 1 2 3 1 0 ↵
List goc: [1][2][3][1][2][3][1]
Nhap k: 1 ↵
List moi: [2][1][3][2][1][3][1]
```

Bài giải: xem trang 315

Bài 218: Thực hiện giải thuật Selection Sort trên danh sách liên kết đơn chứa các trị nguyên, sắp xếp tăng các phần tử chứa trong danh sách. Trong đó, *hoán chuyển các node* thay vì hoán chuyển dữ liệu trong node.

```
Nhap 0 de dung: 1 3 5 7 2 4 6 8 0 ↵
List goc : [1][3][5][7][2][4][6][8]
List tang: [1][2][3][4][5][6][7][8]
```

Bài giải: xem trang 317

Bài 219: Thực hiện giải thuật Bubble Sort trên danh sách liên kết đơn chứa các trị nguyên, sắp xếp tăng các phần tử chứa trong danh sách.

```
Nhap 0 de dung: 1 3 5 7 2 4 6 8 0 ↵
List goc : [1][3][5][7][2][4][6][8]
List tang: [1][2][3][4][5][6][7][8]
```

Bài giải: xem trang 318

Bài 220: Thực hiện giải thuật Insertion Sort trên danh sách liên kết đơn chứa các trị nguyên, sắp xếp tăng các phần tử chứa trong danh sách.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
Nhap 0 de dung: 1 3 5 7 2 4 6 8 0 ↵
List goc : [1][3][5][7][2][4][6][8]
List tang: [1][2][3][4][5][6][7][8]
```

Bài giải: xem trang 319

Bài 221: Xóa một node trong cây BST.

```
Nhap 0 de dung: 9 3 16 14 2 5 4 8 18 0 ↵
2 3 4 5 8 9 14 16 18
Nhap k: 5 ↵
2 3 4 8 9 14 16 18
```

Bài giải: xem trang 320

Bài 222: Thực hiện duyệt cây BST theo chiều sâu (deep-first traversal).

```
Nhap 0 de dung: 9 3 16 14 2 5 4 8 18 0 ↵
LNR: 2 3 4 5 8 9 14 16 18
NLR: 9 3 2 5 4 8 16 14 18
LRN: 2 4 8 5 3 14 18 16 9
```

Bài giải: xem trang 325

Bài 223: Thực hiện thao tác duyệt cây BST theo từng mức (breadth-first traversal).

```
Nhap 0 de dung: 9 3 16 14 2 5 4 8 18 0 ↵
BFS: 9 3 16 2 5 14 18 4 8
```

Bài giải: xem trang 327

Bài 224: Ý nghĩa của từng cách duyệt cây BST, ví dụ minh họa.

```
Nhap 0 de dung: 9 3 16 14 2 5 4 8 18 0 ↵
Cay goc : 9 3 2 5 4 8 16 14 18
Cay copy: 9 3 2 5 4 8 16 14 18
Xoa cay goc... 2 4 5 8 3 14 18 16 9
Cay goc rong
```

Bài giải: xem trang 329

Bài 225: Cho cây BST chứa các trị nguyên. Viết hàm thực hiện hai nhiệm vụ: xuất các node thuộc mức `n` và đếm các node thuộc mức `n`. Gốc có mức `0`.

```
Nhap 0 de dung: 9 3 16 14 2 5 4 8 18 0 ↵
Nhap n: 2 ↵
Muc 2: 2 5 14 18
Tong: 39
```

Bài giải: xem trang 332

Bài 226: Cho cây BST chứa các trị nguyên. Tìm mức của node chứa trị `x` cho trước. Nếu cây không chứa trị `x`, thông báo không tìm thấy. Gốc có mức `0`.

```
Nhap 0 de dung: 9 3 16 14 2 5 4 8 18 0 ↵
Nhap x: 5 ↵
Muc 2
```

Bài giải: xem trang 333

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 227: Cho cây BST chứa các trị nguyên. Nhập các số x, y (x < y, có trong cây BST). Nếu một trong hai node chứa trị x hoặc y là node cha của node kia, trả về node cha. Nếu không, xác định cha chung gần nhất của hai node chứa x và y.

```
Nhap 0 de dung: 9 3 16 14 2 5 4 8 18 0 ↵
Nhap a, b: 4 14 ↵
Node cha: 9
...
Nhap a, b: 16 14 ↵
Node cha: 16
...
Nhap a, b: 4 2 ↵
Node cha: 3
```

Bài giải: xem trang 333

Bài 228: Cho cây BST chứa các trị nguyên. Nhập các số x, y (x < y, có trong cây BST). Xác định đường đi từ node chứa x đến node chứa y.

```
Nhap 0 de dung: 9 3 16 14 2 5 4 8 18 0 ↵
Nhap a, b: 18 5 ↵
5 3 9 16 18
...
Nhap a, b: 2 4 ↵
2 3 5 4
...
Nhap a, b: 9 14 ↵
9 16 14
```

Bài giải: xem trang 334

Bài 229: Thực hiện thao tác chèn một node mới vào cây nhị phân tìm kiếm cân bằng AVL. Tìm kiếm một node trong cây AVL trên.

```
(11,|)
----------
(11,\)
  (4,|)
----------
  (12,|)
(11,|)
  (4,|)
----------
  (12,|)
(11,\)
  (4,\)
    (3,|)
----------
  (12,|)
(11,\)
    (9,|)
  (4,|)
    (3,|)
----------
    (12,|)
  (11,/)
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
(9,|)
    (6,|)
  (4,|)
    (3,|)
----------
Nhap khoa can tim: 12 ↵
[12,Heo]
```

Bài giải: xem trang 335

Bài 230: Thực hiện thao tác xóa một node có trong cây AVL.

```
Cay goc:
    (12,|)
  (11,\)
      (8,|)
    (6,/)
(5,/)
    (4,|)
  (3,/)
Khoa can xoa: 5 ↵
    (12,|)
  (11,|)
    (8,|)
(6,|)
  (4,\)
    (3,|)
```

Bài giải: xem trang 340

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 1: (trang 3)

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

int main() {
  double S;

  printf( "Nhap dien tich S: " );
  scanf( "%lf", &S );
  printf( "The tich V = %g\n",
    ( 4 * M_PI * pow( sqrt( S / ( 4 * M_PI ) ), 3 ) ) / 3 );
  return 0;
}
```

Cách giải thường gặp là tính `R` từ `S`, sau đó dùng `R` tính `V`. Cách này dài dòng vì ta có thể tính `V` trực tiếp theo `S`:

$$\begin{cases} S = 4\pi R^2 \\ V = \dfrac{4}{3}\pi R^3 \end{cases} \Rightarrow V = \frac{4\pi}{3}\left(\sqrt{\frac{S}{4\pi}}\right)^3 \qquad (\pi \approx 3.141593)$$

Bạn phải thực hiện những tính toán sơ bộ như trên trong khi thiết kế chương trình, trước khi cài đặt chương trình bằng một ngôn ngữ lập trình cụ thể.
Định dạng chuỗi `%g` trong `printf()` được dùng khi muốn tối ưu kết quả in một số thực, định dạng này ít được chú ý đến nhưng lại dùng thuận tiện hơn `%f` hoặc `%e`.

Bài 2: (trang 3)

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

int main() {
  float xA, yA, xB, yB;

  printf( "A(xA, yA)? " );
  scanf( "%f%f", &xA, &yA );
  printf( "B(xB, yB)? " );
  scanf( "%f%f", &xB, &yB );
  printf( "|AB| = %g\n",
    sqrt( ( xB - xA ) * ( xB - xA ) + ( yB - yA ) * ( yB - yA ) ) );
  return 0;
}
```

Khi biên dịch dòng lệnh bằng `gcc`, thường gặp lỗi:

```
In function `main': undefined reference to `sqrt'
```

để kết nối đến hàm `sqrt()` trong thư viện `math.h`, cần chỉ định biên dịch như sau:

```
$ gcc c002.c /usr/lib/libm.a -o c002
```

hoặc ngắn gọn hơn:

```
$ gcc c002.c -lm -o c002
```

Bài 3: (trang 3)

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

int main() {
  double xC, yC, R, xM, yM, d;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  printf( "Nhap toa do tam C(xC, yC)? " );
  scanf( "%lf%lf", &xC, &yC );
  printf( "Nhap ban kinh R? " );
  scanf( "%lf", &R );
  printf( "Nhap toa do M(xM, yM)? " );
  scanf( "%lf%lf", &xM, &yM );

  d = R - sqrt( ( xM - xC ) * ( xM - xC ) + ( yM - yC ) * ( yM - yC ) );
  if ( d > 0 ) printf( "M nam trong C()\n" );
  else if ( d < 0 ) printf( "M nam ngoai C()\n" );
       else printf( "M nam tren C()\n" );
  return 0;
}
```

Ta biện luận bằng cách so sánh khoảng cách giữa điểm `M` đến tâm `O` với bán kính `R` của đường tròn. Khoảng cách này được tính từ công thức trong bài 2 (trang 66). Chú ý, nếu bạn nhập các số thực có số ký số phần thập phân lớn hơn độ chính xác của `double`, kết quả sẽ không như mong muốn.

Bài 4: (trang 3)

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

int main() {
  float a, b, c;

  printf( "Nhap 3 canh tam giac: " );
  scanf( "%f%f%f", &a, &b, &c );

  if ( a > 0 && b > 0 && c > 0 && a + b > c && a + c > b && b + c > a ) {
    unsigned f = 0;
    if ( a == b || b == c || c == a ) f += 1;
    if ( a == b && b == c ) f += 1;
    if ( a * a + b * b == c * c || a * a + c * c == b * b || b * b + c * c == a * a )
      f += 3;
    switch ( f ) {
      case 0: printf( "Tam giac thuong\n" ); break;
      case 1: printf( "Tam giac can\n" ); break;
      case 2: printf( "Tam giac deu\n" ); break;
      case 3: printf( "Tam giac vuong\n" ); break;
      case 4: printf( "Tam giac vuong can\n" );
    }
    float p = ( a + b + c ) / 2;
    printf( "Dien tich S = %g\n", sqrt( p * ( p - a ) * ( p - b ) * ( p - c ) ) );
  }
  else printf( "Khong hop le\n" );
  return 0;
}
```

Thông thường bạn phải dùng rất nhiều câu lệnh rẽ nhánh và gặp khó khăn khi xử lý trường hợp tam giác vuông cân. Cách trên tạo cờ hiệu `f` theo cách các lập trình viên C thường dùng, nghĩa là các trị cờ khác nhau sẽ được *cộng thêm* (hoặc `OR`) vào trị cờ rỗng ban đầu. Ví dụ, trong trường hợp tam giác vuông cân: cờ hiệu `f` được cộng thêm `1` khi xét tam giác cân và được cộng thêm `3` khi xét tam giác vuông. Kết quả tam giác

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

vuông cân có `f = 1 + 3 = 4` và tam giác vuông có `f = 0 + 3 = 3` (`0` được gán cho cờ hiệu mặc định, ứng với tam giác thường).

Một cách giải khác: việc sắp xếp lại các cạnh tam giác *theo kích thước* giúp giảm khá nhiều các trường hợp phải xét, nhưng tốn thêm chi phí cho việc sắp xếp. Cách này thường dùng trong đồ họa máy tính, vì việc sắp xếp các cạnh theo kích thước còn tiện dụng cho các giải thuật khác tiếp theo sau.

```
#include <stdio.h>
#include <math.h>
#define swap( a, b ) { float t = a; a = b; b = t; }

int main() {
  float a, b, c;

  printf( "Nhap 3 canh tam giác: " );
  scanf( "%f%f%f", &a, &b, &c );
  /* sắp xếp sao cho: a ≤ b ≤ c */
  if ( a > b ) swap( a, b );
  if ( a > c ) swap( a, c );
  if ( b > c ) swap( b, c );
  if ( a * b * c > 0 && a + b > c ) {
    /* nếu a = c thì a = b */
    if ( a == c ) printf( "Tam giac deu\n" );
    else if ( a == b || b == c )
      /* nếu tam giác vuông, cạnh lớn nhất c là cạnh huyền */
      if ( ( a * a + b * b ) == c * c ) printf( "Tam giac vuong can\n" );
      else                              printf( "Tam giac can\n" );
    else
      if ( ( a * a + b * b ) == c * c ) printf( "Tam giác vuong" );
      else printf( "Tam giac thuong\n" );
    float p = ( a + b + c ) / 2;
    printf( "Dien tich S = %g\n", sqrt( p * ( p - a ) * ( p - b ) * ( p - c ) ) );
  }
  else printf( "Khong hop le\n" );
  return 0;
}
```

Bài 5: (trang 3)

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

double area( double xA, double yA, double xB, double yB, double xC, double yC ) {
  return 0.5 * fabs( xA * yB - xB * yA + xB * yC - xC * yB + xC * yA - xA * yC );
}

int main() {
  double xA, yA, xB, yB, xC, yC, xM, yM;
  double d;

  printf( "A(xA, yA)? " );
  scanf( "%lf%lf", &xA, &yA );
  printf( "B(xB, yB)? " );
  scanf( "%lf%lf", &xB, &yB );
  printf( "C(xC, yC)? " );
  scanf( "%lf%lf", &xC, &yC );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  printf( "M(xM, yM)? " );
  scanf( "%lf%lf", &xM, &yM );

  d = area( xM, yM, xA, yA, xB, yB ) + area( xM, yM, xA, yA, xC, yC )
    + area( xM, yM, xB, yB, xC, yC ) - area( xA, yA, xB, yB, xC, yC );
  if ( d > 0 ) printf( "M nam ngoai tam giac ABC\n" );
  else
    if ( area( xM, yM, xA, yA, xB, yB ) == 0 ||
         area( xM, yM, xA, yA, xC, yC ) == 0 ||
         area( xM, yM, xB, yB, xC, yC ) == 0 )
      printf( "M nam tren canh tam giac ABC\n" );
    else printf( "M nam trong tam giac ABC\n" );
  return 0;
}
```

Khi xác định tam giác bằng tọa độ các đỉnh, không cần phải kiểm tra tam giác có hợp lệ hay không. Tam giác có diện tích bằng 0 cũng hợp lệ, khi đó ba đỉnh của tam giác sẽ thẳng hàng. Công thức tính diện tích tam giác từ tọa độ các đỉnh của nó bằng định thức cấp 3 giúp bài giải ngắn gọn hơn rất nhiều.

Bài 6: (trang 4)

```
#include <stdio.h>

int main() {
  int a, b, c, t;

  printf( "Nhap a, b, c: " );
  scanf( "%d%d%d", &a, &b, &c );
  /* a < b thì hoán chuyển, vậy a ≥ b */
  if ( a < b ) { t = a; a = b; b = t; }
  /* a < c thì hoán chuyển, vậy a ≥ c, kết quả a lớn nhất */
  if ( a < c ) { t = a; a = c; c = t; }
  /* b < c thì hoán chuyển, vậy b ≥ c, kết quả c nhỏ nhất */
  if ( b < c ) { t = b; b = c; c = t; }

  printf( "Tang dan: %d %d %d\n", c, b, a );
  return 0;
}
```

Bạn cũng có thể thiết kế đoạn code hoán chuyển hai trị nguyên dưới dạng macro hoặc hàm:

- Macro `swap()` có tham số: Khi dùng macro, đoạn chương trình được định nghĩa sẽ chèn vào đúng đoạn lệnh gọi macro *trong giai đoạn tiền xử lý* trước lúc biên dịch. Định nghĩa macro:

```
#define swap( a, b ) { int t = a; a = b; b = t; }
```

dùng macro:

```
int a = 5;
int b = 7;
swap( a, b );
```

- Hàm `swap()` dùng con trỏ: Khi dùng hàm, hàm sẽ được gọi tại đoạn lệnh gọi hàm, các tham số sẽ được truyền cho hàm được gọi. Các tham số được truyền cho hàm cần được thay đổi sau khi gọi hàm, nói cách khác hàm sẽ thao tác trên các tham số

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

này và trả lại cho nơi gọi hàm. Vì vậy cần truyền các tham số bằng cách dùng con trỏ. Định nghĩa hàm:

```
void swap( int *a, int *b ) {
  int t = *a;
  *a = *b;
  *b = *t;
}
```

truyền tham số cho hàm:

```
int a = 5;
int b = 7;
swap( &a, &b );
```

Hàm `swap()` dùng tham chiếu chỉ có trong ngôn ngữ C++.
Hoán chuyển `a` và `b` cũng có thể thực hiện mà *không dùng biến phụ*:
`a = a + b; b = a - b; a = a - b;`
hoặc:
`a ^= b ^= a ^= b;`
Tuy nhiên các cách hoán chuyển này chỉ đúng với các trị nguyên và không áp dụng khi hoán chuyển một số với chính nó.

Bài 7: (trang 4)

```
#include <stdio.h>

int main() {
  float a, b;

  printf( "Nhap a, b: " );
  scanf( "%f%f", &a, &b );
  if ( !a ) printf( b ? "Vo nghiem\n" : "Vo dinh\n" );
  else printf( "x = %g\n", -b / a );
  return 0;
}
```

Không có giải thuật phức tạp khi giải quyết bài toán trên. Bạn chỉ cần bảo đảm duyệt hết mọi trường hợp trong lưu đồ sau (lưu đồ này có được bằng cách *biện luận* phương trình bậc nhất):

Chú ý cách dùng biểu thức điều kiện rút gọn, thường dùng trong C:
- Biểu thức điều kiện `( a == 0 )` rút gọn thành biểu thức điều kiện `( !a )`.
- Biểu thức điều kiện `( a != 0 )` rút gọn thành biểu thức điều kiện `( a )`.
Cả hai biểu thức tương đương này đều có bản chân trị (truth table) giống nhau.
Ôn lại cách dùng toán tử 3 ngôi:

```
printf( b ? "Vo nghiem\n" : "Vo dinh\n" );
```

hoặc:

```
b ? printf( "Vo nghiem\n" ) : printf( "Vo dinh\n" );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

nghĩa là:

```
if ( b != 0 ) printf( "Vo nghiem\n" );
else          printf( "Vo dinh\n" );
```

Nên tránh cách dùng toán tử 3 ngôi lồng nhiều lần vì sẽ làm chương trình trở nên khó đọc:

```
!a ? ( !b ? printf( "Vo dinh\n" )
          : printf( "Vo nghiem\n" ) )
   : printf( "x = %g\n", -b/a );
```

Tuy nhiên cách viết này làm nảy sinh một phương pháp giải bài trên rất độc đáo: không dùng `if ... else`, không dùng toán tử 3 ngôi.
Thay "?" bằng "&&", và thay dấu ":" bằng "||", ta nhận được biểu thức điều kiện:

```
#include <stdio.h>
int main() {
  float a, b;

  printf( "Nhap a, b: " );
  scanf( "%f%f", &a, &b );
  !a && ( !b && printf( "Vo dinh\n" ) || printf( "Vo nghiem\n" ) )
     || printf( "x = %g\n", -b/a );
  return 0;
}
```

Do đặc điểm "ngắn mạch" khi định trị biểu thức điều kiện của C, việc định trị biểu thức điều kiện trên cũng rẽ nhánh giống như khi viết chương trình với `if ... else` và bài toán được giải quyết tương tự. Cách giải này chỉ dùng rèn luyện tư duy, ít ý nghĩa thực tế.

Bài 8: (trang 4)

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

int main() {
  float a, b, c;

  printf( "Nhap a, b, c: " );
  scanf( "%f%f%f", &a, &b, &c );

  if ( !a )
    if ( !b ) printf( c ? "Vo nghiem\n" : "Vo dinh\n" );
    else printf( "x = %g\n", -c / b );
  else {
    float d = b * b - 4 * a * c;
    if ( d > 0 )
      printf( "x1 = %g\nx2 = %g\n", ( -b + sqrt( d ) ) / ( 2 * a ),
                                    ( -b - sqrt( d ) ) / ( 2 * a ) );
    else ( !d ) ? printf( "x1 = x2 = %g\n", -b / ( 2 * a ) )
                : printf( "Vo nghiem\n" );
  }
  return 0;
}
```

Giải theo lưu đồ biện luận phương trình bậc hai (xem hình dưới)

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 9: (trang 4)

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

int main() {
  double angle;

  printf( "Nhap so do x cua goc (phut): " );
  scanf( "%lf", &angle );

  angle /= 60;  /* đổi phút thành độ */
  printf( "x thuoc goc vuong thu %d\n", ( int )ceil( angle / 90 ) % 4 );
  printf( "cos(x) = %g\n", cos( angle * M_PI / 180 ) );
  return 0;
}
```

Thay vì phải biện luận kết quả (bằng `if ... else`) để tính số thứ tự góc vuông chứa góc, chương trình trên tính toán từ dữ liệu nhập ra trực tiếp kết quả.
Các hàm lượng giác trong thư viện `<math.h>` của C đều nhận tham số với đơn vị *radian*, 1 radian = degree * π/180. Trong bài trên, cần đổi phút ⇒ độ ⇒ radian.
Bài tập: Xác định tháng `m` thuộc quý nào trong năm (3 quý/năm, 4 tháng/quý).

```
#include <stdio.h>

int main() {
  int m;

  printf( "Nhap thang: " );
  scanf( "%d", &m );
  printf( "Thang %d thuoc quy %d\n", m, ( m - 1 ) / 4 + 1 );
  return 0;
}
```

Bài 10: (trang 5)

```
#include <stdio.h>

int main() {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  unsigned pos, t;
  unsigned long sin;

  while ( 1 ) {
    printf( "SIN (0 de thoat): " );
    scanf( "%lu", &sin );
    if ( !sin ) break;
    unsigned sum = sin % 10;
    sin /= 10;
    for ( pos = 0; pos < 8 && sin > 0; sin /= 10, ++pos ) {
      t = sin % 10;
      if ( pos % 2 ) sum += t;
      else sum += ( 2 * t ) / 10 + ( 2 * t ) % 10;
    }
    printf( "SIN %shop le!\n",
            ( pos < 8 || sin > 0 || sum % 10 ) ? "khong " : "" );
  }
  return 0;
}
```

Để thực hiện bài tập có vẻ phức tạp này, bạn chỉ cần giải quyết được các vấn đề đơn giản sau:
- Lấy chữ số cuối cùng (chữ số phải nhất) của một số `n`: `n % 10`.
- Loại bỏ chữ số cuối cùng của một số `n`: `n /= 10`.
- Cho `d` có nhiều nhất hai chữ số, lấy chữ số hàng chục: `d / 10` và lấy chữ số hàng đơn vị: `d % 10`.

Sau khi tính tổng `sum` (check digit + trọng số), các trường hợp sau là không hợp lệ:
- `pos < 8`: số ký tự của SIN nhỏ hơn 9, điều kiện `sin > 0` ngắt vòng lặp.
- `sin > 0`: số ký tự của SIN lớn hơn 9, điều kiện `pos < 8` ngắt vòng lặp.
- `sum` không chia hết cho 10.

Bài 11: (trang 5)

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

int main() {
  char human, computer;
  unsigned h, c;  /* h: điểm của người; c: điểm của máy */

  srand( time( NULL ) );
  h = c = 0;

  while ( 1 ) {
    printf( "Nhap ky tu (b-d-k), ky tu khac de thoat: " );
    scanf( "%c", &human );
    while ( getchar()  != '\n' ) { }
    switch ( human ) {
      case 'b':
        switch ( rand() % 3 ) {
          case 0: computer= 'b'; break;
          case 1: computer= 'd'; h++; break;
          case 2: computer= 'k'; c++;
        }
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
        break;
      case 'd':
        switch ( rand() % 3 ) {
          case 0: computer= 'b'; c++; break;
          case 1: computer= 'd'; break;
          case 2: computer= 'k'; h++;
        }
        break;
      case 'k':
        switch ( rand() % 3 ) {
          case 0: computer= 'b'; h++; break;
          case 1: computer= 'd'; c++; break;
          case 2: computer= 'k';
        }
        break;
      default: return 0;
    }
    printf( "Computer: %c\n", computer );
    printf( "Ty so: %u - %u\n", h, c );
  }
}
```

Trong C, khi sinh số ngẫu nhiên, chúng ta dùng các hàm sau:

- Hàm `srand()` thiết lập "mầm ngẫu nhiên" (seed) cho loạt số ngẫu nhiên sinh bởi `rand()`. Hàm `srand()` cho phép chương trình dùng `rand()` sinh chuỗi số ngẫu nhiên khác nhau cho mỗi lần chạy do chỉ định điểm khởi đầu khác nhau. Thông thường, để có mầm ngẫu nhiên ta chọn tham số cho `srand()` là hàm `time(NULL)`, thuộc thư viện `time.h`. Hàm này trả về ngày và giờ hiện tại (gọi là *calendar time*) dưới dạng một số nguyên, chính là số giây trải qua từ giây đầu tiên của năm 1900 đến nay.
- Hàm `rand()` dùng sinh một số nguyên ngẫu nhiên giữa `0` và `RAND_MAX` (`32767`). Để sinh số ngẫu nhiên trong nửa đoạn `[0, n)`, dùng `rand() % n`.

Hàm `srand()` và `rand()` hiện được khuyến cáo thay thế do không đủ ngẫu nhiên, đáp ứng bảo mật kém. Tuy nhiên, chúng vẫn dùng tốt với các bài tập trong tài liệu này.

Câu lệnh `fflush( stdin )`, dùng "súc" vùng đệm nhập chuẩn (`stdin`) để loại ký tự `'\n'` còn sót trong lần nhập trước. Nếu không, ký tự này *sẽ được nhận trong lần nhập kế tiếp* làm sai lạc kết quả nhập. Do `fflush()` gọi trên `stdin` có thể có hành vi không xác định trên hệ điều hành Linux nên ta thay dòng lệnh này với:

```
while ( getchar() != '\n' ) { }
```

Đôi lúc cũng dùng `scanf( "%c%*c", &human );` để bỏ qua ký tự `'\n'`.

Bài 12: (trang 5)

```
#include <stdio.h>

int main() {
  float a1, b1, c1, a2, b2, c2, dx, dy, d;

  printf( "Nhap a1, b1, c1: " );
  scanf( "%f%f%f", &a1, &b1, &c1 );
  printf( "Nhap a2, b2, c2: " );
  scanf( "%f%f%f", &a2, &b2, &c2 );

  d = ( a1 * b2 - a2 * b1 );
  dx = ( c1 * b2 - c2 * b1 );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  dy = ( a1 * c2 - a2 * c1 );

  if ( !d ) printf( ( !dx && !dy ) ? "Vo dinh\n" : "Vo nghiem\n" );
  else      printf( "x = %g\ny = %g\n", dx / d, dy / d );
  return 0;
}
```

Người giải thường quên biện luận với các trường hợp hệ phương trình vô nghiệm hoặc vô số nghiệm.

Bài 13: (trang 6)

```
#include <stdio.h>

int main() {
  unsigned d, m, y, top, dayofweek; /* top là số ngày tối đa của tháng */

  printf( "Nhap ngay, thang va nam: " );
  scanf( "%u%u%u", &d, &m, &y );

  if ( y < 1582 ) {
    printf( "Lich Gregorian bat dau tu nam 1582\n" );
    return 1;
  }
  if ( m < 1 || m > 12 ) {
    printf( "Thang khong hop le\n" );
    return 2;
  }
  switch ( m ) {
    case 4: case 6: case 9: case 11: top = 30; break;
    case 2: top = 28 + ( ( y % 4 == 0 && y % 100 ) || y % 400 == 0 ); break;
    default: top = 31;
  }
  if ( d < 1 || d > top ) {
    printf( "Ngay khong hop le\n" );
    return 3;
  }
  printf( "Hop le\n" );
  /* Công thức Zeller */
  y -= ( 14 - m ) / 12;
  m += 12 * ( ( 14 - m ) / 12 ) - 2;
  dayofweek = ( d + y + y / 4 - y / 100 + y / 400 + ( 31 * m ) / 12 ) % 7;
  if ( !dayofweek ) printf( "Chu Nhat\n" );
  else printf( "Thu %u\n", dayofweek + 1 );
  return 0;
}
```

Bài này rèn luyện cách sử dụng cấu trúc `switch` của C, đặc biệt khi xét các trường hợp (case) cho kết quả như nhau.

Cần ghi nhớ rằng trong mỗi phát biểu `case`, phải chấm dứt bằng lệnh `break` để thoát khỏi cấu trúc `switch`, tránh xảy ra việc "đổ xuống" (cascade) phát biểu `case` tiếp. Tuy nhiên, với các trường hợp cho *kết quả như nhau*, việc xét đến phát biểu `case` tiếp lại cần thiết, và khi đó bạn phải xem xét việc có dùng hay không lệnh `break`.

Chúng ta có 3 trường hợp cần xét trong `switch`: 4 tháng có 30 ngày, 7 tháng có 31 ngày và tháng 2 có ngày đặc biệt phụ thuộc năm nhập vào là năm nhuận hay không.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Dễ thấy là nên đặt trường hợp 7 tháng có 31 ngày là trường hợp `default` để chương trình ngắn gọn hơn.

Cấu trúc rẽ nhánh `switch` với một số `case` không có `break`

Thuật toán xét năm nhuận (leap year) được phát biểu như sau: *một năm là năm nhuận nếu nó chia hết cho 4 nhưng không chia hết cho 100, trừ phi nó cũng chia hết cho 400*:

```
( y % 4 == 0 && y % 100 ) || y % 400 == 0
```

Một cách dùng `switch` khác để giải bài tập này, lợi dụng khéo léo tính chất "đổ xuống" khi không có lệnh `break`:

```
top = 29;
switch ( m ) {
  case 1: case 3: case 5: case 7: case 8: case 10: case 12:
          top++;
  /* nếu "đổ xuống" từ đầu dòng trên, đến break, top sẽ là 31
     nếu "đổ xuống" từ đầu dòng dưới, đến break, top sẽ là 30 */
  case 4: case 6: case 9: case 11: top++; break;
  case 2: if ( !( ( y % 4 == 0 && y % 100 ) || y % 400 == 0 ) ) top--;
}
```

Tuy nhiên, không được lạm dụng khả năng "đổ xuống", vì dễ làm chương trình khó hiểu và nhầm lẫn. Ngôn ngữ thừa kế C là C# sau này đã bỏ khả năng đổ xuống dễ nhầm lẫn này.

Bài 14: (trang 6)

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

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
int main() {
  unsigned d, m, y, top;

  printf( "Nhap ngay, thang, nam: " );
  scanf( "%u%u%u", &d, &m, &y );
  switch ( m ) {
    case 4: case 6: case 9: case 11: top = 30; break;
    case 2: top = 28 + ( ( y % 4 == 0 && y % 100 ) || y % 400 == 0 ); break;
    default: top = 31;
  }

  d = ( d % top ) + 1;
  if ( d == 1 ) m = ( m % 12 ) + 1;
  printf( "Ngay mai: %02u/%02u/%u\n", d, m, y + ( m == 1 && d == 1 ) );

  printf( "Nhap ngay, thang, nam: " );
  scanf( "%u%u%u", &d, &m, &y );
  if ( d == 1 ) {
    switch ( m - 1 ) {
      case 4: case 6: case 9: case 11: d = 30; break;
      case 2: d = 28 + ( ( y % 4 == 0 && y % 100 ) || y % 400 == 0 ); break;
      default: d = 31;
    }
    if ( m == 1 ) { m = 12; y--; }
    else m--;
  } else d--;
  printf( "Hom qua: %02u/%02u/%u\n", d, m, y );
  return 0;
}
```

Phát biểu `n % d` ánh xạ trị `n` vào nửa đoạn `[0, d)`. Điều này giúp giảm các phát biểu điều kiện. Ví dụ, thay vì phải viết:

```
if ( d == top ) d = 1;
else d++;
```

nên thay bằng:

```
d = ( d % top ) + 1;
```

Bạn cũng nên chú ý đến chuỗi định dạng `%02u`: in số nguyên không dấu chiếm `2` vị trí, nếu chỉ có một chữ số thì thêm số `0` trước chữ số đó.

Bài 15: (trang 6)

```
#include <stdio.h>

int main() {
  unsigned d, m, y, s, i;

  printf( "Nhap ngay, thang, nam: " );
  scanf( "%u%u%u", &d, &m, &y );

  s = d;
  for ( i = 1; i < m; ++i )
    switch ( i ) {
      case 4: case 6: case 9: case 11: s += 30; break;
      case 2: s += 28 + ( ( y % 4 == 0 && y % 100 ) || y % 400 == 0 ); break;
      default: s += 31;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    }
  printf( "Ngay thu: %u\n", s );
  return 0;
}
```

Bài này là một biến thể của bài 13 (trang 75). Tổng `s` được khởi tạo bằng ngày nhập `d` (cũng là *ngày thứ* `d` trong tháng nhập). Vòng lặp chạy từ tháng 1 đến tháng *trước* tháng nhập. Trong thân vòng lặp, tổng `s` được cộng dồn với *số ngày tối đa* của các tháng phải trải qua trước tháng nhập.

Bạn có thể dùng hàm `difftime()` của `time.h`, nếu hiểu rõ cách thao tác với thông tin thời gian trong C, xem bài 191 (trang 279):

```
int days_diff( unsigned d, unsigned m, unsigned y ) {
  time_t first, now;
  struct tm date = { 0 };
  date.tm_year = y - 1900;  /* ngày đang xét */
  date.tm_mon  = m - 1;
  date.tm_mday = d;
  now = mktime ( &date );
  date.tm_mon  = 0;         /* ngày đầu năm */
  date.tm_mday = 1;
  first = mktime ( &date );
  return ( int ) difftime( now, first ) / (24 * 60 * 60);
}

/* dùng hàm trên trong hàm main() */
printf( "Ngay thu: %d\n", days_diff( d, m, y ) + 1 );
```

Bài 16: (trang 7)

```
#include <stdio.h>

int main() {
  unsigned y, m, dow, d, top, y1, m1;

  printf( "Nhap nam: " );
  scanf( "%u", &y );
  /* Công thức Zeller, tính thứ cho ngày đầu năm (ngày 1/1) */
  m = 1;
  y1 = y - ( 14 - m ) / 12;
  m1 = m + 12 * ( ( 14 - m ) / 12 ) - 2;
  dow = ( 1 + y1 + y1 / 4 - y1 / 100 + y1 / 400 + ( 31 * m1 ) / 12 ) % 7;
  /* lặp cho 12 tháng */
  for ( m = 1; m <= 12; ++m ) {
    printf( "Thang %u\n", m );
    switch ( m ) {
      case 4: case 6: case 9: case 11: top = 30; break;
      case 2: top = 28 + ( ( y % 4 == 0 && y % 100 ) || y % 400 == 0 ); break;
      default: top = 31;
    }
    printf( "  S  M  T  W  T  F  S\n" );
    for ( d = 0; d < dow; ++d )
      printf( "    " );
    for ( d = 1; d <= top; ++d ) {
      printf( "%3u", d );
      if ( ( dow + d ) % 7 == 0 ) putchar( '\n' );
    }
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    dow = ( dow + top ) % 7; /* tính lại thứ cho ngày đầu mỗi tháng */
    if ( dow ) putchar( '\n' );
  }
  return 0;
}
```

Có hai vấn đề cần giải quyết trong bài này:

- Thứ của ngày đầu tiên mỗi tháng:

Trước hết, tính thứ của ngày đầu tiên của năm `dow` bằng công thức Zeller. Thứ của ngày đầu tiên tháng sau: `(dow + top) % 7` (chú ý `dow ∈ [0, 7)`) với `top` là số ngày tối đa của tháng trước.

- Xuống hàng sau khi in một hàng lịch: nếu ngày đầu tháng đều là Chúa nhật thì đơn giản.

```
for ( d = 1; d <= top; ++d ) {
  printf( "%3u", d );
  if ( d % 7 == 0 ) putchar( '\n' );
}
```

Tuy nhiên, ta phải in ký tự `space` nếu chưa đến thứ của ngày đầu tiên trong tháng (`dow`). Nghĩa là điều kiện để xuống dòng bây giờ là `(dow + d) % 7 == 0`.

Sau khi in lịch một tháng, ta cần xuống dòng để in lịch tháng kế tiếp. Nhưng nếu `(dow + d) % 7 == 0` ta không xuống dòng để tránh xuống dòng hai lần, do kết thúc một dòng lịch trùng với kết thúc in lịch một tháng.

Do `dow` mới cũng bằng `(dow + top) % 7`, ta tính `dow` mới trước rồi mới kiểm tra để chèn ký tự xuống dòng.

```
dow = (dow + top) % 7;
if ( dow ) putchar('\n');
```

Bài này cũng rèn luyện cách dùng phối hợp nhiều vòng lặp.

Bài tập: Linux cung cấp hai lệnh dùng in lịch, lệnh `cal` in lịch theo chiều ngang và lệnh `ncal` in lịch theo chiều dọc (xem hình dưới). Viết chương trình thực hiện hai lệnh này. Chương trình nhận dữ liệu vào là tháng và năm, dùng công thức Zeller để tính thứ của ngày đầu tiên trong tháng.

```
Thang va nam: 2 2016 ↵
2/2016
'cal' command:
  S  M  T  W  T  F  S
     1  2  3  4  5  6
  7  8  9 10 11 12 13
 14 15 16 17 18 19 20
 21 22 23 24 25 26 27
 28 29

'ncal' command:
Thang 12
  S     7 14 21 28
  M  1  8 15 22 29
  T  2  9 16 23
  W  3 10 17 24
  T  4 11 18 25
  F  5 12 19 26
  S  6 13 20 27
```

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

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
#include <math.h>

int main() {
  unsigned y, m, d, dow, top, y1, m1, col, i, j;
  char s[] = { 'S', 'M', 'T', 'W', 'T', 'F', 'S' };
  printf( "Thang va nam? " );
  scanf( "%u %u", &m, &y );
  y1 = y - ( 14 - m ) / 12;
  m1 = m + 12 * ( ( 14 - m ) / 12 ) - 2;
  dow = ( 1 + y1 + y1 / 4 - y1 / 100 + y1 / 400 + ( 31 * m1 ) / 12 ) % 7;
  switch (m) {
    case 4: case 6: case 9: case 11: top = 30; break;
    case 2: top = 28 + ( ( y % 4 == 0 && y % 100 ) || y % 400 == 0 ); break;
    default: top = 31;
  }
  printf( "%u/%u\n", m, y );
  printf( "'cal' command:\n" );
  for ( i = 0; i < 7; ++i ) printf( "%3c", s[i] );
  putchar( '\n' );
  for ( d = 0; d < dow; ++d ) printf( "   " );
  for ( d = 1; d <= top; ++d ) {
    printf( "%3u", d );
    if ( ( dow + d ) % 7 == 0 ) putchar( '\n' );
  }
  if ( ( dow + top ) % 7 ) putchar('\n');
  printf("\n'ncal' command:\n" );
  col = ( unsigned ) ceil( ( dow + top ) / 7.0 );
  for ( i = 0; i < 7; ++i, putchar( '\n' ) )
    for ( d = i + 1 - dow, j = 0; j <= col; ++j )
      if ( j == 0 ) printf( "%3c", s[i] );
      else if ( j == 1 && i < dow ) printf( "   " );
      else if ( ( d += j > 1 ? 7 : 0 ) <= top ) printf( "%3u", d );
      else break;
  return 0;
}
```

Bài 17: (trang 7)

```
#include <stdio.h>

int main() {
  unsigned y, m, dow, s, i, p, top, d;

  printf( "Nhap nam: " );
  scanf( "%u", &y );
  printf( "Nhap thu cho ngay dau tien cua nam: " );
  scanf( "%u", &dow );
  printf( "Nhap thang: " );
  scanf( "%u", &m );
  /* ngày đầu tháng m là ngày thứ s trong năm */
  s = 0;
  for ( i = 1; i <= m; ++i, s += top )
    switch ( i ) {
      case 4: case 6: case 9: case 11: top = 30; break;
      case 2: top = 28 + ( ( y % 4 == 0 && y % 100 ) || y % 400 == 0 ); break;
      default: top = 31;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    }
  /* top hiện là số ngày của tháng m, vì vậy s phải trừ bớt top */
  s -= top;
  /* thứ tự người trực p vào ngày đầu tháng m */
  for ( p = 0, i = dow; i < s + dow; ++i )
    /* có 5 người trực và Chúa nhật không trực */
    if ( i % 7 ) p = ( p + 1 ) % 5;
  /* thứ dow của ngày đầu tháng m */
  dow = i % 7; /* tương đương dow = ( s + dow ) % 7 */
  printf( "    Sun    Mon    Tue    Wen    Thu    Fri    Sat\n" );
  for ( d = 0; d < dow; ++d )
    printf( "%7c", ' ' );
  for ( d = 1; d <= top; ++d ) {
    printf( "%3u", d );
    if ( ( dow + d - 1 ) % 7 ) {
      printf( " [%c]", p["ABCDE"] );
      p = ( p + 1 ) % 5;
    } else printf( " [ ]" );
    if ( ( dow + d ) % 7 == 0 ) putchar( '\n' );
  }
  if ( ( dow + top ) % 7 ) putchar( '\n' );
  return 0;
}
```

Đây là bài tập tổng hợp kiến thức có được khi giải các bài tập từ 13 - 16.
- Ngày đầu tiên của tháng quan tâm (`m`) là ngày thứ `s` trong năm. Ta cũng cần số ngày (`top`) của tháng `m`, nên vòng lặp tính cả tháng `m`. Vì vậy kết quả `s` phải bớt đi `top`.
- Từ thứ đầu tiên `dow` cho đến `s + dow`, nếu không phải Chúa nhật (`i % 7 == 0`), người trực thứ `p` tăng nhưng vẫn trong đoạn `[0, 5)`. Xác định được người trực ngày đầu tháng `m`.
- Dễ dàng xác định thứ của ngày đầu tháng `m`: `dow = (s + dow) % 7`.
Sau đó ta có thể in lịch tháng kèm theo người trực.

Bài 18: (trang 7)

```
#include <stdio.h>

int main() {
  long h;

  printf( "Nhap so gio: " );
  scanf( "%ld", &h );
  printf( "%ld tuan, %ld ngay, %ld gio\n",
       h / ( 24 * 7 ), ( h % ( 24 * 7 ) ) / 24, ( h % ( 24 * 7 ) ) % 24 );
  return 0;
}
```

Bài tập rèn luyện cách dùng các toán tử `/` và `%`.

Bài 19: (trang 8)

```
#include <stdio.h>

int main() {
  int h, m, s;
  long time;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  printf( "Nhap gio, phut, giay [1]: " );
  scanf( "%d%d%d", &h, &m, &s );
  time = 3600 * h + 60 * m + s;

  printf( "Nhap gio, phut, giay [2]: " );
  scanf( "%d%d%d", &h, &m, &s );
  time -= 3600 * h + 60 * m + s;
  if ( time < 0 ) time = -time;

  printf( "Hieu thoi gian: %ld gio, %ld phut, %ld giay\n",
          time / 3600, ( time % 3600 ) / 60, ( time % 3600 ) % 60 );
  return 0;
}
```

Khi cần thao tác trên dữ liệu có nhiều *đơn vị tính toán* khác nhau, ta phải chuyển dữ liệu về một đơn vị tính toán chung.

Bài 20: (trang 8)

```
#include <stdio.h>

int main() {
  unsigned kw;
  unsigned long money;

  printf( "Nhap so kW tieu thu: " );
  scanf( "%u", &kw );
  /* cứ tính giá sàn, tiền phụ thu tính sau */
  money = kw * 500;
  /* từ kw 100 trở đi, thêm phụ thu (800 - 500) cho mỗi kw tăng thêm */
  if ( kw > 100 ) money += ( kw - 100 ) * 300;
  /* từ kw 250 trở đi, thêm phụ thu (1000 - 800) cho mỗi kw tăng thêm */
  if ( kw > 250 ) money += ( kw - 250 ) * 200;
  /* từ kw 350 trở đi, thêm phụ thu (1500 - 1000) cho mỗi kw tăng thêm */
  if ( kw > 350 ) money += ( kw - 350 ) * 500;
  printf( "Chi phi: %lu\n", money );
  return 0;
}
```

Có nhiều bài tập tương tự như tính tiền nước, tiền điện (tính phụ thu khi vượt chỉ tiêu), tính tiền phòng, tiền taxi (có chiết khấu khi sử dụng nhiều).
Cách tính thông thường sẽ đơn giản chuyển yêu cầu của bài tập thành các phát biểu rẽ nhánh tương ứng. Cách tính này dài dòng, dễ nhầm lẫn và có thể gây tràn các biến trung gian khi tính toán:

```
#include <stdio.h>

int main() {
  unsigned kw;
  unsigned long money;

  printf( "Nhap so kW tieu thu: " );
  scanf( "%u", &kw );

  if ( kw <= 100 ) money = kw * 500;
  else if ( kw <= 250 ) money = 500 * 100 + 800 * ( kw - 100 );
  else if ( kw <= 350 )
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    money = 500 * 100 + 800 * ( 250 - 100 ) + 1000 * ( kw - 250 );
  else
    money = 500 * 100 + 800 * ( 250 - 100 ) +
            1000 * ( 350 - 250 ) + 1500 * ( kw - 350 );
  printf( "Chi phi: %lu\n", money );
  return 0;
}
```

Phương pháp tốt nhất là tính toàn bộ chi phí với *mức phí cơ bản* (giá sàn). Sau đó tùy theo điều kiện mà tính tiếp phần chi phí được giảm (hoặc tăng) cho thích hợp. Xem các chú giải chi tiết trong bài giải chính.

Bài 21: (trang 8)

```
#include <stdio.h>

int main() {
  float sd, d1, d2, d3;
  char zone;
  unsigned beneficiary;

  printf( "Nhap diem chuan: " );
  scanf( "%f", &sd );
  printf( "Nhap diem 3 mon thi: " );
  scanf( "%f%f%f", &d1, &d2, &d3 );
  while ( getchar() != '\n' ) { }
  printf( "Nhap khu vuc (A, B, C, X): " );
  scanf( "%c", &zone );
  printf( "Nhap doi tuong (1, 2, 3, 0): " );
  scanf( "%u", &beneficiary );

  if ( d1 * d2 * d3 ) {
    float d = d1 + d2 + d3;
    switch ( zone ) {
      case 'A': d += 2; break;
      case 'B': d += 1; break;
      case 'C': d += 0.5;
    }
    switch ( beneficiary ) {
      case 1: d += 2.5; break;
      case 2: d += 1.5; break;
      case 3: d += 1;
    }
    printf( "%s [%g]\n",  d >= sd ? "Dau" : "Rot", d );
  }
  else printf( "Rot (co mon diem 0)\n" );
  return 0;
}
```

Để kiểm tra trong 3 điểm nhập vào `d1`, `d2` và `d3` có điểm `0` hay không, xét tích `d1 * d2 * d3`.

Bài 22: (trang 8)

```
#include <stdio.h>

int main() {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  unsigned n, i, count, sum;

  printf( "Nhap n: " );
  scanf( "%u", &n );

  printf( "Cac uoc so: " );
  for ( count = sum = 0, i = 1; i <= n; ++i )
    if ( n % i == 0 ) {
      printf( "%u ", i );
      count++;
      sum += i;
    }

  printf ( "\nCo %u uoc so, tong la: %u\n", count, sum );
  return 0;
}
```

Bài tập này rèn luyện một kỹ năng thường dùng: cách thu thập thông tin qua các vòng lặp:
- Một biến tích lũy (`count`, `sum`, …) *phải* được khởi tạo *trước khi* vào vòng lặp, thay đổi trong thân vòng lặp (tăng dần từng đơn vị, cộng tích lũy, …) rồi được hiển thị *sau khi* chấm dứt vòng lặp.

Bài 23: (trang 8)

```
#include <stdio.h>

int main() {
  unsigned n, i, j, sum;

  printf( "Nhap n: " );
  scanf( "%u", &n );

  printf( "Cac so hoan hao nho hon %u: ", n );
  for ( i = 1; i < n; ++i ) {
    for ( sum = 0, j = 1; sum <= i && j < i; ++j )
      if ( i % j == 0 ) sum += j;
    if ( sum == i ) printf( "%u ", i );
  }
  putchar( '\n' );
  return 0;
}
```

Thêm điều kiện (`sum <= i`) cho vòng lặp bên trong sẽ giảm đáng kể số vòng lặp cần xét.

Bài 24: (trang 9)

```
#include <stdio.h>

int main() {
  unsigned long n, t;
  unsigned u, sum = 0, count = 0;

  printf( "Nhap n: " );
  scanf( "%lu", &n );
  t = n;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  do {
    u = t % 10;
    count++;
    sum += u;
  } while ( t /= 10 );
  printf( "%lu co %u chu so\n", n, count );
  printf( "Chu so cuoi cung la: %lu\n", n % 10 );
  printf( "Chu so dau tien la: %u\n", u );
  printf( "Tong cac chu so la: %u\n", sum );
  printf( "So dao nguoc la: " );
  while ( n % 10 == 0 ) n /= 10;  /* loại các số 0 đầu số đảo */
  do printf( "%lu" , n % 10 ); while ( n /= 10 ) ;
  putchar( '\n' );
  return 0;
}
```

Việc kết hợp vừa định trị vừa dùng làm biểu thức điều kiện rất phổ biến khi dùng vòng lặp `while`, `do ... while`, bạn cần rèn luyện cách viết này. Ví dụ:

```
do {
  printf( "%u" , n % 10 );
} while ( n /= 10 );
```

Giải thích: sau khi `n` chia cho `10`, nếu `n` còn khác `0` (nghĩa là `n < 10`) thì vòng lặp vẫn còn hoạt động.

Biểu thức điều kiện của C sẽ được định trị để xác định đúng sai: trị bất kỳ khác `0` là true (đúng), trị bằng `0` là false (sai). C nạp trị này vào thanh ghi và kiểm tra các bit để xác định đúng sai.

Điều này dẫn đến một lỗi rất thường gặp với người mới làm quen C là nhầm lẫn giữa dấu "==" của phép so sánh bằng với dấu "=" của phép gán. Ví dụ:

```
if ( i = 0 ) printf( "i bang 0\n" );
else         printf( "i khac 0\n" );
```

Kết quả luôn xuất: "`i khac 0`" với mọi `i` nhập vào, kể cả khi nhập `i = 0`.

Người viết đã nhầm lẫn biểu thức kiểm tra với phép gán. Trình biên dịch sẽ gán `i` bằng `0`, rồi kiểm tra biểu thức điều kiện. Vì biểu thức này bằng `0` nên luôn cho kết quả false và phần `else` của cấu trúc `if ... else` sẽ được thực hiện.

Lập trình viên có kinh nghiệm thường dùng mẹo sau để tránh lỗi: thay vì bạn viết: `if ( i == 0 ) ...` , dùng: `if ( 0 == i )...`

Trong trường hợp nhầm lẫn dấu `==` với `=`:

`if ( i = 0 ) ...` : chương trình vẫn chạy và cho kết quả sai.

`if ( 0 = i ) ...` : sẽ báo lỗi do *không thể gán cho một hằng*. Lỗi này gọi là lỗi `Lvalue`: trị bên trái dấu `=` được C cho là một "modifiable lvalue", nghĩa là phải thay đổi được. Ngôn ngữ thừa kế C là C# cũng chú ý loại bỏ đặc tính dễ nhầm lẫn này.

Do thứ tự các yêu cầu nên bài được triển khai như trên. Tuy nhiên, cũng có cách thực hiện khác.

Bài tập: Nhập vào một số tự nhiên `n` (có nhiều hơn 2 chữ số), cho biết `n` có bao nhiêu chữ số, tìm 2 chữ số đầu tiên và 2 chữ số cuối cùng của `n`.

```
Nhap n: 12345 ↵
12345 co 5 chu so: [12] .. [45]
```

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

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
int main() {
  unsigned long n;
  int c;

  printf( "Nhap n: " );
  scanf( "%lu%n", &n, &c );
  printf( "%lu co %d chu so: [%lu] .. [%lu]\n",
          n, c, n / ( unsigned ) pow( 10, c - 2 ), n % 100 );
  return 0;
}
```

Bài 25: (trang 9)

```
#include <stdio.h>

int main() {
  unsigned a, b, gcd, lcm;

  printf( "Nhap cap (a, b): " );
  scanf( "%u%u", &a, &b );

  gcd = a;
  while ( a % gcd || b % gcd ) gcd--;
  printf( "USCLN (a, b): %u\n", gcd );
  lcm = a;
  while ( lcm % a || lcm % b ) lcm++;
  printf( "BSCNN (a, b): %u\n", lcm );
  return 0;
}
```

Có nhiều giải thuật cho bài này trình bày trong nhiều sách lập trình, chủ yếu là giải thuật tính USCLN. Chúng tôi giới thiệu 2 giải thuật phổ biến:

- Dùng giải thuật vét cạn (cách trên). Gọi `gcd` là USCLN, khởi tạo `gcd` bằng `a` hoặc `b` (tốt nhất bằng số nhỏ hơn). Sau đó trừ `gcd` dần từng *đơn vị* cho đến khi `a` và `b` *đều* chia hết cho `gcd`, `gcd` cuối sẽ là USCLN.

Như vậy điều kiện chấm dứt vòng lặp là `a` và `b` đều chia hết cho `gcd`:

`( a % gcd == 0 && b % gcd == 0 )`

Suy ra, điều kiện để vòng lặp tồn tại là phủ định mệnh đề trên, theo định lý De Morgan[18] là:

`( a % gcd != 0 || b % gcd != 0 )`

và viết tắt: `( a % gcd || b % gcd )`

- Dùng thuật toán Euclid:

```
#include <stdio.h>

int main() {
  unsigned a, b, c;

  printf( "Nhap cap (a, b): " );
  scanf( "%u%u", &a, &b );
  c = a * b;
  while ( a - b ) ( a > b )? ( a -= b ) : ( b -= a );
  printf( "USCLN (a, b): %u\n", a );
```

[18] Augustus De Morgan (1806 - 1871)

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  printf( "BSCNN (a, b): %u\n", c/a );
  return 0;
}
```

+ Chú ý biểu thức điều kiện `(a - b)` của vòng lặp `while` tương đương biểu thức điều kiện `(a != b)` và trước khi tiến hành giải thuật tính USCLN phải tính trước tích `a * b` vì sau giải thuật `a`, `b` đã thay đổi.

+ Nếu bạn viết:

```
while ( a - b ) ( a > b ) ? a -= b : b -= a;
```

bạn có thể sẽ nhận kết quả sai do quá trình tối ưu mã đối tượng tùy theo trình biên dịch. Bạn nên viết với cặp ngoặc đơn bao quanh:

```
while ( a - b ) ( a > b ) ? ( a -= b ) : ( b -= a );
```

Ta cũng gặp điều tương tự khi dùng macro. Có vài nguyên tắc đơn giản để tránh các *hiệu ứng lề* do macro gây ra: dùng cặp ngoặc đơn bao quanh maro và các tham số của macro, không nên truyền một biểu thức như là một tham số của macro. Ví dụ:

```
#define Cube( x ) x * x * x
```

sẽ hoạt động sai với: `Cube( x + 1 )`, vì trình biên dịch sẽ hiểu là:

```
x + 1 * x + 1 * x + 1 tức: x + (1 * x) + (1 * x) + 1.
```

Ta cần viết như sau:

```
#define Cube(x) ( (x) * (x) * (x) )
```

+ Thuật toán tính USCLN trong cách trên không chính xác với `a` (hoặc `b`) âm hoặc bằng `0`. Chúng được suy không đầy đủ từ thuật toán Euclid như nhiều sách lập trình mô tả. Thuật toán Euclid dùng công thức truy hồi (recurrence) sau:

$$\begin{cases} \text{gcd}(0, b) = b \\ \text{gcd}(a, b) = \text{gcd}(b \text{ MOD } a, a) \quad (a > 0) \end{cases}$$

Vì vậy thuật toán Euclid có thể thực hiện dễ dàng bằng đệ quy:

```
unsigned gcd( unsigned a, unsigned b ) {
  return ( !a ) ? b : gcd( b % a, a );
}
```

Chú ý:

- USCLN(0, 0) không xác định, ta không xét ở đây.
- BSCNN(a, b) có thể tính được từ USCLN(a, b).

Bài tập: Nhập một số thực (kiểu `double`) dương, phần thập phân có tối đa 6 ký tự. Tách số nhập vào thành phần nguyên và phần thập phân rồi in ra màn hình:

```
Nhap mot so float: 123.456789 ↵
Phan nguyen    : 123
Phan thap phan: 456789
```

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

int main() {
  double n, m;
  long t;
  int i = 0;

  printf( "Nhap mot so double: " );
  scanf( "%lf", &n );
  printf( "Phan nguyen    : %ld\n", t = ( long ) n );
  do {
    i++;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    m = n * pow( 10, i ) - t * pow( 10 , i );
  } while ( n * pow( 10, i ) - ( long ) ( t * pow( 10 , i ) + m ) );
  printf( "Phan thap phan: %ld\n", ( long ) m );
  return 0;
}
```

Gọi `m` là trị float của phần thập phân của số thực `n`.
Ý tưởng là số thực (`n`) và phần nguyên (`t`) của nó không bao giờ bằng nhau (`n - t != 0`) khi phần thập phân của số thực còn khác `0`. Để phần thập phân của số thực là `0`, ta nhân số thực `n` với `10` một số lần, cũng đúng bằng số lần nhân `10` để chuyển `m` thành trị nguyên. Tuy nhiên, do hạn chế về độ chính xác của kiểu `double`, khi so sánh `n` và `t` ta phải "đẩy" chúng sang phần nguyên rồi mới so sánh.
Nếu chỉ đơn giản lấy phần nguyên và phần thập phân, dùng hàm `modf()` trong thư viện `math.h`.

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

int main() {
  double n, fractpart, intpart;
  n = 9.123456;
  fractpart = modf(n, &intpart);
  printf("Phan nguyen   : %lf\n", intpart);
  printf("Phan thap phan: %lf\n", fractpart);
  return 0;
}
```

Bài 26: (trang 9)

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

int main() {
  int num, denom;
  int tnum, tdenom;

  printf( "Nhap tu so, mau so: ");
  scanf( "%d%d", &num, &denom );

  tnum = abs( num );
  tdenom = abs( denom );

  while ( tnum != tdenom )
    ( tnum > tdenom ) ? ( tnum -= tdenom ) : ( tdenom -= tnum );
  /* tnum bây giờ bằng gcd( num, denom ) */
  num /= tnum;
  denom /= tnum;

  printf( "Rut gon: " );
  if ( denom < 0 ) {
    num = -num;
    denom = -denom;
  }
  if ( denom == 1 ) printf( "%d\n", num );
  else              printf( "%d/%d\n", num, denom );
  return 0;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
}
```

Thuật toán Euclid tính USCLN áp dụng trong bài 25 (trang 86), chỉ đúng với hai số *nguyên dương*, vì vậy cần lấy giá trị tuyệt đối của hai số trước khi tiến hành thuật toán.

Để phân số có dạng in thuận tiện, ta biện luận như sau:

Nếu tử số và mẫu số khác dấu, xét mẫu số; nếu mẫu số `< 0`, ta đổi dấu cả tử số và mẫu số để dấu `-` chuyển sang tử số.

Nếu tử số và mẫu số cùng dấu, xét mẫu số; nếu mẫu số `< 0` (nghĩa là tử số cũng `< 0`), ta đổi dấu cả tử số và mẫu số để loại dấu `-`.

Mô tả điều kiện trên bằng ngôn ngữ lập trình:

```
if ( num * denom < 0 ) {
  if ( denom < 0 ) { num = -num; denom = -denom; }
} else {
  if ( denom < 0 ) { num = -num; denom = -denom; }
}
```

Ta rút gọn thành:

```
if ( denom < 0 ) { num = -num; denom = -denom; }
```

Bài 27: (trang 9)

```
#include <stdio.h>

int main() {
  unsigned n, i = 2;

  printf( "Nhap n: " );
  scanf( "%u", &n );
  while ( n > 1 ) {        /* lặp khi n > 1 */
    while ( n % i ) ++i;   /* tìm ước số i đầu tiên > 1 của n */
    /* chia n cho i vừa tìm được và phân tích thừa số nguyên tố tiếp */
    n /= i;
    /* i chính là ước số nguyên tố nhỏ nhất của n */
    printf( n > 1 ? "%u * " : "%u\n", i );
  }
  return 0;
}
```

Trước hết bạn hãy quan sát cách tính thừa số nguyên tố của `n` trong hình dưới:

- Bên trái ta chia `n` cho số nguyên tố vừa tìm được bên phải, và thực hiện điều này cho đến khi `n = 1` (nghĩa là lặp trong khi `n > 1`).
- Bên phải ta tìm *ước số nguyên tố nhỏ nhất* với mỗi `n` tương ứng bên trái.
- Kết quả cột bên trái (các ước số nguyên tố nhỏ nhất tìm được) là các thừa số nguyên tố nhận được khi phân tích `n` thành thừa số nguyên tố.

n / i	i (không cần khởi tạo)
12	2
6	2
3	3
1	

12 = 2 * 2 * 3

Ta nhận xét: ước số `i > 1` nhỏ nhất của một số `n` *luôn là một số nguyên tố*. Bởi vì nếu `i` không phải là số nguyên tố thì sẽ tìm được một ước số `> 1` của `n` nhỏ hơn `i`, đó là ước số của chính `i`. Như vậy, ở phía bên phải, thay vì phải tìm *ước số nguyên tố nhỏ nhất* của `n` ta chỉ cần tìm *ước số > 1 nhỏ nhất* của `n`. Nói cách khác, phía bên phải chỉ cần tìm *ước số > 1 đầu tiên* của `n` bên trái.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Chú ý, vòng lặp tìm ước số lần sau bắt đầu từ `i` cũ.
Một cách giải khác:
- Tìm các ước số của `n` lớn hơn `1`.
- Nếu tìm thấy *loại bỏ thừa số nguyên tố đó* (`n /= i`), in nó ra và tìm tiếp.

Vì một thừa số có thể xuất hiện nhiều lần, dùng `while` thay vì `if`.

```
for ( i = 2; i <= n; ++i )
  while ( n % i == 0 ) {
    n /= i;
    printf( n > 1 ? "%u * " : "%u\n", i );
  }
```

Có thể dùng giải thuật đệ quy nhưng phức tạp hơn nhiều:

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

void PrimeAnalyse( unsigned prime, unsigned n ) {
  if ( prime > ( unsigned )( sqrt( n ) ) ) {
    printf( "%u\n", n );
    return;
  }
  if ( n % prime == 0 ) {
    printf( "%u * ", prime );
    PrimeAnalyse( prime, n/prime );
    return;
  }
  else PrimeAnalyse( prime + 1, n );
}

int main() {
  unsigned n;
  printf( "Nhap n: " );
  scanf( "%u", &n );
  PrimeAnalyse( 2, n );
  return 0;
}
```

Đây là một bài toán đệ quy kiểu "chặn trên", từng số `prime` được chuyển đến hàm `PrimeAnalyse()` để xác định có phải là ước số của `n` không, cho đến khi `prime` vượt quá điểm "chặn trên" $\sqrt{n}$ (điểm dừng của đệ quy). Trong hàm, khi một `prime` được xác định là ước số (`n % prime == 0`) thì `prime` đó được in ra và kiểm tra tiếp với `n/prime` xem còn là ước số không, nếu không `prime` mới (`prime + 1`) sẽ được kiểm tra.
Để giảm số lần đệ quy, dòng: `PrimeAnalyse( prime + 1, n );`
được viết thành:

```
if ( prime == 2 ) PrimeAnalyse( 3, n );
else              PrimeAnalyse( prime + 2, n );
```

Bài 28: (trang 10)

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

int main() {
  double x;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  long round;
  unsigned n;

  printf( "Nhap so thuc x: ");
  scanf( "%lf", &x );
  printf( "Do chinh xac: ");
  scanf( "%u", &n );

  /* x = x * 10^n, mục tiêu dịch chuyển phần cần
     làm tròn thành phần nguyên của số thập phân */
  x *= pow( 10, n );

  /* tiến hành làm tròn, chuyển kiểu tự động thành long */
  if ( x > 0 ) round = ( long )( x + 0.5 );
  else         round = ( long )( x - 0.5 );
  /* x = x / 10^n, ngược với dịch chuyển lúc đầu */
  x = round / pow( 10, n );

  printf( "%.*lf\n", n > 0 ? n : 0 , x );
  return 0;
}
```

Chuỗi định dạng: `%.*lf` có ý nghĩa như sau:

```
x = 3.141593;
n = 4;
printf( "%.*lf", n, x );
```

Ký tự `*` cho biết *số ký tự dành chỗ* được chỉ định bởi một tham số (kiểu `int`) của `printf()` tại vị trí tương ứng.
Nghĩa là:
`printf( "%.*lf\n", n , x );`
Với `n = 4`, tương đương với: `printf( "%.4lf\n", x );`

Bài 29: (trang 10)

```
#include <stdio.h>

int main() {
  unsigned cel, fah;

  printf( "%10s%14s\n", "Celcius", "Fahrenheit" );
  for ( cel = 0; cel <= 10; ++cel )
    printf( "%10u%14.2f\n", cel, 9.0 * cel / 5 + 32 );
  printf( "\n%10s%14s\n", "Fahrenheit", "Celcius" );
  for ( fah = 32; fah <= 42; ++fah )
    printf( "%10u%14.2f\n", fah, 5.0 * ( fah - 32 ) / 9 );
  return 0;
}
```

Bài 30: (trang 10)

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
#include <stdio.h>

int main() {
  double rate, balance;
  unsigned year, period, n;

  do {
    printf( "Nhap lai suat, tien von, thoi han: " );
    n = scanf( "%lf,%lf,%u", &rate, &balance, &period );
    while ( getchar() != '\n' ) { }
    if ( n != 3 || n == EOF )
      printf( "Nhap thieu hoac nhap loi!\n" );
  } while ( n != 3 || n == EOF );

  printf( "Lai suat: %g%%\n", rate * 100 );
  printf( "Von ban dau: %g\n", balance );
  printf( "Thoi han: %u nam\n", period );
  printf( "%3s%10s\n", "Nam", "Von" );
  for ( year = 1; year <= period; ++year ) {
    balance *= 1 + rate;
    printf( "%3u%10g\n", year, balance );
  }
  return 0;
}
```

Bài này dùng vòng lặp `do while` để kiểm soát nhập với hàm `scanf()`: hàm `scanf()` sẽ trả về số phần tử được gán trị thành công. Nếu xuất hiện lỗi `scanf()` trả về trị `EOF`. Một ví dụ về dùng hàm `scanf()` có kiểm tra lỗi và có chú ý loại bỏ ký tự `'\n'` còn sót lại trong vùng đệm nhập:

```
printf( "Nhap mot so integer: " );
while ( scanf( "%d", &x ) != 1 ) {
  while ( getchar() != '\n' ) { }
  printf( "Nhap mot so integer: " );
}
```

Hàm `scanf()` còn được sử dụng rất linh hoạt để xử lý dữ liệu nhập, xem thêm bài 174 (trang 254).

Bài 31: (trang 11)

```
#include <stdio.h>

int main() {
  unsigned i, j;

  printf( "Bang cuu chuong\n" );
  for ( i = 1; i <= 10; ++i ) {
    for ( j = 2; j <= 9; ++j )
      printf( "%c%2ux%2u=%2u", 179, j, i, i * j );
    printf( "%c\n", 179 );
  }
  return 0;
}
```

In từng dòng cho tất cả các bảng cửu chương rồi mới chuyển sang in dòng kế tiếp. Các bảng cửu chương cách nhau bởi ký tự có mã `ASCII 179`.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 32: (trang 11)

```
#include <stdio.h>

int main() {
  char c;
  do {
    unsigned n;
    printf( "Nhap n: " );
    scanf( "%u", &n );
    while ( getchar() != '\n' ) { }
    unsigned num = 0, i = 0;
    while ( 1 ) {
      printf( "%5u", n );
      if ( n == 1 && num > 1 && ++num ) break;
      if ( n % 2 ) n = n * 3 + 1;
      else n /= 2;
      num++; i++;
      if ( i % 6 == 0 ) putchar( '\n' );
    }
    printf( "\nHailstones sinh duoc: %u\n", num );
    do {
      printf( "Tiep (y/n)? ");
      scanf( "%1[^\n]c", &c );
      while ( getchar() != '\n' ) { }
    } while ( c != 'y' && c != 'n' );
  } while ( c != 'n' );
  return 0;
}
```

Khó khăn chủ yếu của bài tập này là xác định các Hailstones của `1`, nguyên do là trị `1` cũng được xem như *điều kiện để kết thúc* vòng lặp. Tuy nhiên, ta dễ dàng phân biệt trị `1` này là trị đưa vào để tính hay trị cuối chuỗi Hailstones dựa vào việc *đếm số phần tử* của chuỗi Hailstones được sinh ra:

```
if ( n == 1 && num > 1 && ++num ) break;
```

Nếu hai điều kiện đầu là true thì tăng `num` và do `num > 0` nên cũng true.
Có vài cách thể hiện vòng lặp vĩnh viễn trong C như:

```
for ( ; ; ) { }
while ( 1 ) { }
```

Để thoát khỏi các vòng lặp vĩnh viễn này, dùng phát biểu `break`, `goto` hoặc `return`, nhưng lập trình viên có kinh nghiệm thường không dùng `goto`.

Bài 33: (trang 11)

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

int main() {
  unsigned i, j, s;

  printf( "So Amstrong co 3, 4 chu so:\n" );
  /* i dùng duyệt các số có 3, 4 chữ số */
  for ( i = 100; i < 9999; ++i ) {
    /* j dùng tách các chữ số của i ra để kiểm tra */
    for ( s = 0, j = i; s <= i && j > 0; j /= 10 )
      s += ( unsigned )pow( j % 10, i < 1000 ? 3 : 4 );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    if ( i == s ) printf( "%u ", i );
  }
  putchar( '\n' );
  return 0;
}
```

Tương tự bài 23 (trang 84), thêm điều kiện `sum <= i` cho vòng lặp bên trong sẽ giảm đáng kể số vòng lặp cần xét.

Một cách giải khác, dùng macro thay cho dùng hàm `pow()`:

```
#include <stdio.h>
#define p3( x ) x * x * x
#define p4( x ) x * x * x * x

int main() {
  unsigned i, j, k, l, d;

  for ( i = 1; i < 10; ++i )
    for ( j = 0; j < 10; ++j )
      for ( k = 0; k < 10; ++k ) {
        d = 100*i + 10*j + k;
        if ( p3(i) + p3(j) + p3(k) == d )
          printf( "%u ", d );
        for ( l = 0; l < 10; ++l )
          if ( p4(i) + p4(j) + p4(k) + p4(l) == 10*d + l )
            printf( "%u ", 10*d + l );
      }
  return 0;
}
```

Bài 34: (trang 11)

```
#include <stdio.h>
#include <math.h>
#define eps 1e-6

double f( double x ) {
  return pow( sin( x ), 2 ) * cos( x );
}

double inter( double a, double b, unsigned n ) {
  unsigned i;
  double h = ( b - a )/n;
  double t = 0.5 * ( f( a ) + f( a + n * h ) );
  for ( i = 1; i < n; ++i )
    t += f( a + i * h );
  return t * h;
}

int main() {
  unsigned n = 10;
  double a = 0;
  double b = 1.5708;        /* π/2 */
  double t, t1;

  t1 = inter( a, b, n );
  do {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    t = t1;
    n *= 2;
    t1 = inter( a, b, n );
  } while ( fabs( t1 - t ) / 3 > eps );

  printf ( "Ket qua  : %lf\n", t1 );
  printf ( "Doi chung: %lf\n", pow( sin( b ), 3 ) / 3 );
  return 0;
}
```

Công thức hình thang dùng tính tích phân xác định: với $h = \frac{b-a}{n}$, $x_i = a + ih$

$$\int_a^b f(x)dx \approx h\left[\frac{f(x_0)+f(x_1)}{2} + \frac{f(x_1)+f(x_2)}{2} + \ldots + \frac{f(x_{n-1})+f(x_n)}{2}\right]$$

$$= h\sum_{i=0}^{n}\frac{f(x_i)+f(x_{i+1})}{2} = h\sum_{i=0}^{n}\frac{f(a+ih)+f(a+(i+1)h)}{2}$$

biểu thức cuối ở trên đã thuận lợi cho việc tổ chức vòng lặp chạy từ `0` đến `n - 1`, thân vòng lặp là tổng tích lũy của 1/2 tổng *hai lời gọi hàm*:

```
double inter( double a, double b, unsigned n ) {
  unsigned i;
  double h = ( b - a )/n;
  double t = 0.0;

  for ( i = 0; i < n; ++i )
    t += ( f( a + i * h ) + f( a + ( i + 1 ) * h ) ) / 2;
  return t * h;
}
```

nhưng biểu thức sau giúp dễ xây dựng vòng lặp hơn và cũng chạy nhanh hơn (thân vòng lặp chỉ gọi hàm một lần):

$$h\left[\frac{f(x_0)+f(x_1)}{2} + \frac{f(x_1)+f(x_2)}{2} + \ldots + \frac{f(x_{n-1})+f(x_n)}{2}\right]$$

$$= h\left[\frac{f(x_0)+f(x_n)}{2} + f(x_1) + \ldots + f(x_{n-1})\right]$$

$$= h\left[\frac{f(x_0)+f(x_n)}{2} + \sum_{i=1}^{n-1} f(x_i)\right] = h\left[\frac{f(a)+f(a+nh)}{2} + \sum_{i=1}^{n-1} f(a+ih)\right]$$

```
double inter( double a, double b, unsigned n ) {
  unsigned i;
  double h = ( b - a )/n;
  double t = 0.5 * ( f( a ) + f( a + n * h ) );

  for ( i = 1; i < n; ++i )
    t += f( a + i * h );
  return t * h;
}
```

Bạn cần nhận thấy mối tương quan nhất định của các ký hiệu toán học Π hoặc Σ với vòng lặp `for`. Những biến đổi sơ bộ công thức toán học làm giai đoạn lập trình trở nên thuận tiện hơn.

Bài 35: (trang 12)

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

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
int isPrime( unsigned k ) {
  unsigned i;
  if ( k == 2 ) return 1;
  if ( k < 2 || k % 2 == 0 ) return 0;
  for ( i = 3; i < ( k >> 1 ); i += 2 )
    if ( k % i == 0 ) return 0;
  return 1;
}

int main() {
  unsigned n;

  printf( "Nhap n: " );
  scanf( "%u", &n );

  if ( isPrime( n ) )
    printf( "%u la so nguyen to\n", n );
  else {
    printf( "%u khong la so nguyen to\n", n );
    unsigned i;
    for ( i = n - 1; i >= 2; --i )
      if ( isPrime( i ) ) {
        printf( "So nguyen to be hon gan nhat: %u\n", i );
        break;
      }
  }
  return 0;
}
```

Một số nguyên tố `k` là số nguyên lớn hơn `1`, chỉ chia hết cho `1` và cho chính nó. Bạn phải kiểm tra điều này bằng cách lần lượt kiểm tra các số từ `2` (đương nhiên `k` chia hết cho `1`) đến `k` xem `k` có chia hết cho số nào không. Nếu số cần kiểm tra chưa đến `k` mà `k` đã chia hết cho nó thì `k` không phải là số nguyên tố.

Thực tế chỉ cần kiểm tra từ 2 đến $\sqrt{k}$: điều kiện vòng lặp là $k < \sqrt{n}$, hoặc $k * k < n$. Theo bất đẳng thức Cauchy[19]: $\sqrt{k} \leq \frac{(k + 1)}{2}$, vì vậy thay vì dùng $\sqrt{k}$ (phải có thư viện `math.h`) và tính toán chậm hơn, ta có thể dùng:

$(k + 1)/2 = k/2 + 0.5 \approx k/2 + 1$

Với số nguyên, phép chia cho 2^n có thể thay thế bằng phép dịch phải bit `>> n` đơn vị. Ví dụ: thay vì kiểm tra với `1 + k/2`, ta kiểm tra với `1 + (k >> 1)`.

Phép dịch bit được thực thi nhanh hơn do phần cứng hỗ trợ.

Bạn cũng có thể lấy số nguyên tố gần nhất, bé hơn n, bằng cách đệ quy như sau. Cách này thú vị do gần với tư duy của con người:

```
#include <stdio.h>

int isPrime( unsigned k ) {
  unsigned i;
  if ( k == 2 ) return 1;
  if ( k < 2 || k % 2 == 0 ) return 0;
  for ( i = 3; i < (k >> 1); i += 2 )
    if ( k % i == 0 ) return 0;
```

[19] Augustin Louis Cauchy (1789 - 1857)

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  return 1;
}

int getPrime( unsigned n ) {
  if ( isPrime( n ) ) return n;
  return getPrime( n - 1 );
}

int main() {
  unsigned n, t;

  printf( "Nhap n (n > 2): " );
  scanf( "%u", &n );

  t = getPrime( n );
  if ( t == n )
    printf( "%u la so nguyen to\n", n );
  else {
    printf( "%u khong la so nguyen to\n", n );
    printf( "So nguyen to be hon gan nhat: %u\n", t );
  }
  return 0;
}
```

Bài 36: (trang 12)

```
#include <stdio.h>

int main() {
  unsigned n, k, i, count;

  printf( "Nhap n: " );
  scanf( "%u", &n );

  k = 2;
  count = 0;
  while ( count < n ) { /* vòng lặp kiểm tra k có phải là số nguyên tố */
    for ( i = 2; i * i <= k; ++i )
      if ( k % i == 0 ) break;
    if ( i * i > k ) {
      printf( "%u ", k );
      count++;
    }
    k++;
  }
  putchar( '\n' );
  return 0;
}
```

Vòng lặp duyệt các số `i` từ `2` đến $\sqrt{k}$, nếu `k` chia được cho một trong các số đó dùng lệnh `break` để thoát khỏi vòng lặp. Như vậy có hai trường hợp kết thúc vòng lặp, xác định bởi trị của `i` khi kết thúc vòng lặp:

- Nếu `i <=` $\sqrt{k}$, nghĩa là `k` có ước số và không phải là số nguyên tố.
- Nếu `i >` $\sqrt{k}$, nghĩa là `k` đã vượt qua vòng lặp kiểm tra và là số nguyên tố.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 37: (trang 13)

```
#include <stdio.h>

int main() {
  unsigned n, m, s;

  do {
    printf( "Nhap n: " );
    scanf( "%u", &n );
  } while ( !n && printf( " Error: n > 0\n" ) );

  s = 0;
  m = 1;
  while ( s + m < n ) {
    printf( "%u", m );
    s += m++;
    if ( s + m < n ) printf( " + " );
  }
  if ( s ) {
    printf( " = %u < %u\n", s, n );
    printf( "m = %u\n", m - 1 );
  }
  else printf( "Khong tim thay\n" );
  return 0;
}
```

Chú ý một cách báo lỗi của vòng lặp nhập số:

```
do {
  printf( "Nhap n: " );
  scanf( "%u", &n );
} while ( n <= 0 && printf( " Error: n > 0\n" ) );
```

Hàm `printf()` trả về số ký tự xuất, trong trường hợp trên vế phải biểu thức điều kiện luôn đúng.

- Trong trường hợp nhập sai (`n <= 0`), kiểm tra vế trái biểu thức điều kiện cho kết quả đúng nên tiếp tục kiểm tra vế phải và lời báo lỗi được in ra.
- Trong trường hợp nhập đúng (`n > 0`), kiểm tra vế trái biểu thức điều kiện cho kết quả sai; do tính chất "ngắn mạch" của biểu thức điều kiện, nên không cần kiểm tra biểu thức vế phải nữa.

Nếu chỉ tìm `m`, có thể viết ngắn gọn như sau:

```
#include <stdio.h>

int main() {
  unsigned n, m, s;

  do {
    printf( "Nhap n: " );
    scanf( "%u", &n );
  } while ( n <= 0 && printf( " Error: n > 0\n" ) );
  s = 0;
  m = 1;
  while( s + m < n ) s += m++;
  if ( s ) printf( "m = %u\n", m - 1 );
  else printf("Khong tim thay\n");
  return 0;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
}
```

Bài 38: (trang 13)

```
#include <stdio.h>

int main() {
  unsigned x, y, z, x1, y1, z1, min, n;

  printf( "Nhap n (nghin dong, n > 5): " );
  scanf( "%u", &n );

  x = y = z = x1 = y1 = z1 = 0;
  min = n;
  for ( y = n / 2; y > 0; --y )
    for ( z = 0; z < n / 5; ++z ) {
      x = ( n - 2 * y - 5 * z );
      if ( ( y > x + z ) && ( x + y + z < min ) ) {
        min = x + y + z;
        x1 = x;  y1 = y;  z1 = z;
      }
    }

  printf( "( %u, %u, %u ): %u\n", x1, y1, z1, min );
  return 0;
}
```

Bài tập này thực chất là giải hệ phương trình;

$\begin{cases} x + 2y + 5z = n \\ y > x + z \end{cases}$ với $(x + y + z)$ nhỏ nhất

bằng cách thử tất cả tập hợp nghiệm có thể.

Các bài tập từ 38 - 42 dùng nhiều vòng lặp lồng nhau để duyệt "vét cạn" một tập hợp lớn các khả năng cần xét.

Hai biện pháp được áp dụng để giảm số vòng lặp lồng nhau, đưa đến giảm kích thước tập hợp cần xét:

- Từ $x + 2y + 5z = n$, suy ra $2y \le n$ và $5z < n$, nên tập hợp cần xét thu nhỏ lại: $y \in [1, \frac{n}{2}]$ và $z \in [0, \frac{n}{5})$. Vì $y > x + z$ nên ta thiết kế vòng lặp theo y chạy từ lớn đến nhỏ để có kết quả nhanh hơn.
- Từ $x + 2y + 5z = n$, suy ra $x = n - (2y + 5z)$, ta loại bỏ được vòng lặp theo x.

Bộ kết quả (x, y, z) được lưu vào (x1, y1, z1), đồng thời tổng $x + y + z$ (số tờ bạc của đáp án) được tính để so sánh với lần sau nhằm thu được bộ kết quả có số tờ bạc ít nhất.

Bài 39: (trang 13)

```
#include <stdio.h>

int main() {
  unsigned x, y, z;

  for ( x = 1; x < 100; ++x )
    for ( y = 1; y < 100; ++y )
      for ( z = 1; z < 100; ++z )
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
        if ( x * x + y * y == z * z ) {
          if ( y - x == 1 && z - y == 1 )
            printf( "(%u, %u, %u): ba so nguyen lien tiep\n", x, y, z );
          if ( y % 2 == 0 && y - x == 2 && z - y == 2 )
            printf( "(%u, %u, %u): ba so chan lien tiep\n", x, y, z );
        }
  return 0;
}
```

Chú ý các điều kiện:

- 3 số nguyên liên tiếp a, b, c: `b - a = c - b = 1`.

Không viết sai biểu thức điều kiện như sau: `if ( b - a == c - b == 1 )...`

Biểu thức điều kiện trên có cú pháp hợp lệ nhưng (`c - b == 1`) sẽ được định trị trước (`0` hoặc `1`), sau đó (`b - a`) sẽ so sánh với kết quả định trị này dẫn đến sai.

- 3 số chẵn liên tiếp a, b, c: `b - a = c - b = 2`, b (hoặc a, c) chẵn.

Bài 40: (trang 13)

```
#include <stdio.h>

int main() {
  unsigned x, y, z;

  for ( x = 1; x < 100 / 5; ++x )
    for ( y = 1; y < 100 / 3; ++y ) {
      z = 100 - ( x + y );
      if ( 300 == 15 * x + 9 * y + z )
        printf( "( %2u, %2u, %2u )\n", x, y, z );
    }
  return 0;
}
```

Bài tập này thực chất là giải hệ phương trình:

$$\begin{cases} x + y + z = 100 \\ 5x + 3y + \frac{1}{3}z = 100 \end{cases} \Leftrightarrow \begin{cases} x + y + z = 100 \\ 15x + 9y + z = 300 \end{cases}$$

bằng cách thử tất cả tập hợp nghiệm có thể.

Hai biện pháp được áp dụng để giảm số vòng lặp lồng nhau và giảm kích thước tập hợp cần xét:

- Từ $5x + 3y + z / 3 = 100$, suy ra $5x < 100$ và $3y < 100$, nên tập hợp cần xét thu nhỏ lại: $x \in [1, \frac{100}{5})$ và $y \in [1, \frac{100}{3})$
- Từ $x + y + z = 100$, suy ra $z = 100 - (x + y)$, ta loại bỏ được vòng lặp theo z.

Bài 41: (trang 13)

```
#include <stdio.h>

float f( float x, float y, int d ) {
  switch ( d ) {
    case 0: return ( float ) x + y;
    case 1: return ( float ) x - y;
    case 2: return ( float ) x * y;
    case 3: return ( float ) x / y;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  }
  return 0;
}

int main() {
  unsigned i, j, k, l, m;
  char sign[] = { '+', '-', '*', '/' };

  for ( i = 0; i < 4; ++i )
   for ( j = 0; j < 4; ++j )
    for ( k = 0; k < 4; ++k )
     for ( l = 0; l < 4; ++l )
      for ( m = 0; m < 4; ++m )
       if ( f( f( f( f( f(1, 2, i), 3, j), 4, k), 5, l), 6, m) == 36 )
        printf( "((((1 %c 2) %c 3) %c 4) %c 5) %c 6 = 36\n",
                sign[i], sign[j], sign[k], sign[l], sign[m] );
  return 0;
}
```

Có 5 dấu ? (5 phép toán có thể) và 4 toán tử; nên cần 5 vòng lặp lồng nhau, mỗi vòng lặp duyệt qua 4 toán tử để "vét cạn" tất cả khả năng có thể có.
Do không xác định được toán tử sẽ sử dụng, biểu thức cần tính được tổ chức thành dạng gọi hàm, thuận lợi cho việc thể hiện yêu cầu trong biểu thức điều kiện hơn. Trong đó, các toán tử (+, -, *, /) tương ứng (0, 1, 2, 3), được thay bằng tham số truyền đến hàm.
Các phép toán là như nhau và có thể lặp lại, nên ta không tối ưu các vòng lặp.

Bài 42: (trang 14)

```
#include <stdio.h>

int isPrime( unsigned k ) {
  unsigned i;
  if ( k == 2 ) return 1;
  if ( k < 2 || k % 2 == 0 ) return 0;
  for ( i = 3; i < (k >> 1); i += 2 )
    if ( k % i == 0 ) return 0;
  return 1;
}

int main() {
  unsigned p[200], i, n, count;
  unsigned x, y, z;

  /* mảng dùng kiểm chứng chứa các số nguyên tố 1 < n < 1000
     nhưng chỉ kiểm chứng các số nguyên tố 5 < n < 1000 */
  for ( n = 0, i = 2; i < 1000; ++i )
    if ( isPrime( i ) ) p[n++] = i;
  /* n - 3: trừ 3 số nguyên tố < 5 (2, 3, 5) */
  printf( "Co %u so nguyen to n (5 < n < 1000)\n", n - 3 );
  for ( count = 0, i = 3; i < n; ++i ) {
    for ( x = 0; x < i; ++x )
      for ( y = x; y < i; ++y )
        for ( z = i; z > y; --z )
          if (p[i] == p[x] + p[y] + p[z]) {
            printf("%3u = %u + %u + %u\n", p[i], p[x], p[y], p[z]);
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
            count++;
            goto checked;
          } /* end if */
    checked: { }
  } /* end for i */
  printf( "Da kiem tra %u so nguyen to\n", count );
  return 0;
}
```

Phát biểu nhảy `goto` được khuyến cáo là không nên dùng vì sẽ tạo mã không cấu trúc và phức tạp (spaghetti code). Tuy nhiên phát biểu `goto` rất hữu hiệu khi cần thoát ra hàng loạt vòng lặp lồng nhau như trường hợp bài tập trên. Phát biểu `break` chỉ có thể giúp thoát khỏi *một vòng lặp* đang chứa nó.
Nếu không muốn dùng `goto`, đưa thân của vòng lặp `i` vào một hàm.
Với `n` lớn, chương trình chạy chậm (ví dụ `n = 10000`, phải kiểm chứng đến `1226` số nguyên tố). Chú ý sau khi vòng lặp `x` chọn, vòng lặp `y` chỉ chọn số nguyên tố từ vị trí `x` trở đi, tương tự với `z`. Nhận xét này giúp giảm số lần lặp xuống nhiều lần.

Bài 43: (trang 14)

```
#include <stdio.h>

int main() {
  unsigned n, i, Fi, Fi1, Fi2;

  printf( "Nhap n (n < 40): " );
  scanf( "%ld", &n );

  Fi = Fi1 = Fi2 = 1;
  for ( i = 3; i <= n; ++i ) {
    Fi = Fi1 + Fi2;
    Fi1 = Fi2;
    Fi2 = Fi;
  }
  printf( "Fi(%u) = %u\n", n, Fi );
  return 0;
}
```

Bài tập này điển hình cho cách tính "dịch chuyển", thường gặp khi tính biểu thức truy hồi tuyến tính:

Để tính `Fi` mới, các trị liên quan (`Fi1` và `Fi2`) cần được cập nhật, "dịch chuyển" lên cho đồng bộ, thực hiện bằng cách gán dồn các trị lên.
Cách khác, khó hiểu hơn một chút:

```
unsigned n, i, Fi, Fi1, t;
/* ... */
Fi1 = Fi = 1;
for ( i = 3; i <= n; ++i ) {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  t = Fi;        /* lưu lại Fi cũ */
  Fi += Fi1;     /* Fi mới = Fi cũ + Fi1 đứng trước Fi */
  Fi1 = t;       /* cập nhật Fi1 bằng Fi cũ */
}
printf( "Fi(%u) = %u\n", n, Fi );
```

hoặc dùng mảng:

```
unsigned n, i, Fi[40];
/* ... */
Fi[1] = Fi[2] = 1;
for ( i = 3; i <= n; ++i )
  Fi[i] = Fi[i-1] + Fi[i-2];
printf( "Fi(%u) = %u\n", n, Fi[n] );
```

Tham khảo thêm thuật toán tính nhanh số Fibonnaci bằng ma trận lũy thừa, bài 93 (trang 155).

Bài 44: (trang 14)

```
#include <stdio.h>

int main() {
  unsigned n, i, j;

  printf( "Nhap n: " );
  scanf( "%u", &n );

  for ( i = 1; i <= n; ++i, putchar( '\n' ) )
    for ( j = 1; j <= n + i - 1; ++j )
      printf( ( j < n - i + 1 )? "  " : "* " );

  for ( i = 1; i <= n; ++i, putchar( '\n' ) )
    for ( j = 1; j <= n + i - 1; ++j )
      printf( ( j == n - i + 1 || j == n + i - 1 || i == n ) ? "* " : "  " );
  return 0;
}
```

Bài này phổ biến trong các sách bài tập lập trình, thường được xem như bài tập rèn luyện cách tính toán trong vòng lặp một cách trực giác. Vấn đề đơn giản hơn nhiều nếu nhìn dưới khía cạnh toán học.

Bản chất của yêu cầu trên là vẽ các điểm trên màn hình theo tọa độ. Bạn hãy xem mô tả đồ thị của bài trên trong hình dưới, chú ý là trục tung `i` trên màn hình quay xuống dưới, gốc tọa độ là (1, 1).

Đường `j = n + i - 1` là tịnh tiến của đường `j = n` và đường `j = n - i + 1` là đối xứng qua trục `j = n` của đường `j = n + i - 1`.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Để vẽ cả dòng lẫn cột bạn dùng 2 vòng lặp lồng vào nhau: vòng lặp ngoài tính dòng (biến đếm `i`), với mỗi dòng, vòng lặp trong vẽ các cột (biến đếm `j`). Hai vòng lặp này quét tất cả các vị trí phải vẽ (ký tự `*` hoặc ký tự `space`).
Khi vẽ tam giác đặc, bạn quét toàn bộ miền:

$$\begin{cases} 1 \le i \le n \\ 1 \le j \le n + i - 1 \end{cases}$$

trong đó: với $1 \le j < n - i + 1$ ta vẽ dấu `space`
với $n - 1 + 1 \le j \le n + i - 1$, tức phần còn lại, ta vẽ dấu `*`

Khi vẽ tam giác rỗng, có ba trường hợp vẽ `*` tương ứng với phương trình ba cạnh, dễ dàng xác định các trường hợp này dựa vào đồ thị trên.
Bài tập: Áp dụng cách suy luận trên để vẽ hai tam giác đặc, đối đỉnh, mỗi tam giác có chiều cao là `n` nhập từ bàn phím.

```
#include <stdio.h>

int main() {
  unsigned n, i, j;

  printf( "Nhap n: " );
  scanf( "%u", &n );

  for ( i = 1; i <= 2 * n - 1; ++i, putchar( '\n' ) )
    for ( j = 1; j <= 2 * n - 1; ++j )
      if ( ( j >= i && j <= 2 * n - i ) || ( j <= i && j >= 2 * n - i ) )
        printf( "* " );
      else printf( "  " );
  return 0;
}
```

Bài 45: (trang 14)

```
#include <stdio.h>

int main() {
  unsigned n, i, j;

  printf( "Nhap n (n < 5): " );
  scanf( "%u", &n );

  for ( i = 1; i <= 2 * n - 1; ++i, putchar( '\n' ) )
    for ( j = 1; j <= 2 * n - 1; ++j )
      if ( ( j <= i && j <= 2 * n - i ) || ( j >= i && j >= 2 * n - i ) )
        printf( "%2u", j );
      else printf( "  " );
  return 0;
}
```

Tính toán như bài 44, nhưng thay vì vẽ dấu `*` ta in trị `j`, nghĩa là in *trị của chỉ số cột* tương ứng.

Bài 46: (trang 14)

```
#include <stdio.h>

int main() {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  unsigned long n, s, i;

  printf( "Nhap n: " );
  scanf( "%lu", &n );

  for ( s = 0, i = 1; i <= n; ++i )
    s += i * i * i;
  printf( "Ve trai = %lu\n", s );
  printf( "Ve phai = %lu\n", n * n * ( n + 1 ) * ( n + 1 ) / 4 );
  return 0;
}
```

Bài 47: (trang 15)

```
#include <stdio.h>

int main() {
  unsigned n, i, s;

  printf( "Nhap n: " );
  scanf( "%u", &n );
  for ( s = 0, i = n; i > 0; i -= 2 ) s += i;
  printf( "S = %u\n", s );
  return 0;
}
```

Khác nhau giữa cách tính hai tổng là *trị khởi đầu*, có thể dùng biểu thức điều kiện để tính (dựa vào `n` chẵn hay lẻ):

```
for ( s = 0, i = ( n % 2 ) ? 1 : 2; i <= n; i += 2 )
  s += i;
```

Tổng chẵn có thể bắt đầu bằng `0`, như vậy: `i = ( n % 2 ) ? 1 : 0;`
chú ý `1` và `0` cũng là định trị đúng hoặc sai cho biểu thức điều kiện (`n % 2`), nên ta có thể viết rút gọn: `i = ( n % 2 );`
và vòng lặp trở thành:

```
for ( s = 0, i = n % 2; i <= n; i += 2 )
  s += i;
```

Bạn nên chú ý cách viết này. Một ví dụ, hàm kiểm tra xem `n` có là số lẻ không,

thay vì viết:

```
int isOdd( int n ) {
  if ( n % 2 ) return 0;
  return 1;
}
```

nên viết như sau:

```
int isOdd( int n ) {
  return ( n % 2 );
}
```

Ta nhận thấy hai tổng trên đều kết thúc với `n`, nên nếu ta chọn trị khởi đầu là `n` và tính ngược lại, cách tính hai tổng trên không khác nhau, cách giải đơn giản hơn.

```
for ( s = 0, i = n; i > 0; i -= 2 ) s += i;
```

Bài 48: (trang 15)

```
#include <stdio.h>

int main() {
  unsigned i, n, temp;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  printf( "Nhap n: " );
  scanf( "%u", &n );

  for ( i = n / 2; i >= 1; --i )
    if ( n % i == 0 && i & 1 ) break;
  printf( "US le lon nhat: %u\n", i );

  for ( temp = i = 1; i <= n / 2; i *= 2 )
    if ( n % i == 0 ) temp = i;

  printf( "US lon nhat la luy thua cua 2: %u\n",temp );
  return 0;
}
```

Hai yêu cầu của bài tập có vẻ giống nhau nhưng cách tiến hành lại rất khác nhau:
- Với câu a) ta tiến hành theo cách thông thường là lấy từng ước số của `n` kiểm tra xem có chia hết cho 2 (ước số lẻ) không. Để nhanh chóng lấy được ước số lớn nhất, ta duyệt các số theo chiều ngược từ `n/2` (ước số lớn nhất có thể) đến `1` (ước số nhỏ nhất có thể), và ngắt vòng lặp duyệt khi phát hiện ra ước số lẻ đầu tiên.
- Với câu b) khó có thể tiến hành như trên là lấy từng ước số của `n` kiểm tra xem có phải là lũy thừa của 2 không, do thuật toán kiểm tra lũy thừa của 2 không dễ. Ý tưởng ở đây là suy nghĩ với thứ tự ngược lại: lấy từng lũy thừa của 2 trong đoạn cần tìm kiểm tra xem có phải là ước số của `n` không. Điều này thực hiện được vì ta có thể chủ động tạo ra lần lượt tất cả các số là lũy thừa của 2 trong đoạn cho phép, bắt đầu từ lũy thừa của 2 nhỏ nhất ($2^0 = 1$). Phải có một biến tạm lần lượt lưu các ước số tìm được (ước số tìm được sau sẽ chồng lên ước số trước), và sau khi kết thúc việc duyệt các lũy thừa của 2 (nghĩa là khi lũy thừa của 2 đang xét lớn hơn `n/2`) ước số còn nằm lại trong biến lưu tạm sẽ là ước số lớn nhất.

Khi kiểm tra `i` phải là số lẻ hay không, ta dùng điều kiện:

```
if ( i % 2 ) printf( "Odd number" );
```

như vậy nghĩa là ta đã tư duy theo hệ thập phân: `i` chia cho 2 cho số dư khác 0. Còn một cách kiểm tra khác:

```
if ( i & 1 ) printf( "Odd number" );
```

Ta đã tư duy theo hệ nhị phân: dùng toán tử `AND` để kiểm tra xem bit cuối của `i` có khác 0 hay không. Toán tử `AND` được thực hiện nhanh hơn.

Với sự giúp sức của hàm `pow()` (thư viện `math.h`) bạn có thể tính câu b) cách khác, tuy nhiên không hay bằng cách trên:

```
for ( temp = i = 1; pow( 2, i ) <= n / 2; ++i )
  if ( n % i == 0 ) temp = i;
printf( "US lon nhat la luy thua cua 2: %u\n", temp );
```

Trong trường hợp bài này, câu b) cũng có thể giải theo cách của câu a), do chúng ta có cách xác định nhanh một số có phải là lũy thừa của 2 hay không.

Do những tính chất đặc biệt của toán học số bù 2 (two's complement), ta viết được macro kiểm tra một số có phải là lũy thừa của 2 hay không:

```
#define ispow2(x) (!((x) & ((x)-1)) && (x))
```

Macro này thực chất kiểm tra xem bit MSB có phải là bit duy nhất bằng 1 không:

`x`	`1000`	bit MSB bằng `1` với `x` là lũy thừa của 2
`x - 1`	`0111`	`x - 1` là mặt nạ của `x` dùng kiểm tra bit MSB
`x & (x - 1)`	`0000`	
`ispow2( x )`	`1111`	

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Ngoài ra x phải khác 0;
Câu b) được viết lại như sau:

```
#define ispow2(x) (!((x) & ((x)-1)) && (x))
for ( i = n / 2; i >= 1; --i )
  if ( n % i == 0 && ispow2( i ) ) break;
printf( "US lon nhat la luy thua cua 2: %u\n", i );
```

Cách này tuy đáng chú ý nhưng vẫn không nhanh hơn cách được dùng trong bài giải trên.

Bài 49: (trang 15)

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

int main() {
  unsigned n, k;
  double S = 1.0;

  printf( "Nhap n: " );
  scanf( "%u", &n );
  for ( k = 2; k <= n; ++k )
    S = pow( k + S, 1.0 / ( k + 1 ) );
  printf( "Ket qua: %g\n", S );
  return 0;
}
```

Xem xét biểu thức:

$$S = \sqrt[n+1]{n + \sqrt[n]{(n-1) + \sqrt[n-1]{(n-2) + \ldots + \sqrt[3]{2 + \sqrt{1}}}}}$$

Dễ dàng nhận thấy công thức truy hồi của căn liên tục trên:

$S_k = \sqrt[k+1]{k + S_{k-1}}, k = 1..n$

Điều kiện đầu: $S_1 = \sqrt{1} = 1$

Như vậy, ta cần thực hiện một vòng lặp với biến đếm k chạy từ 2 (vì S_1 ứng với k = 1 đã được tính) đến n. Biểu thức trong thân vòng lặp:

$S_k = \sqrt[k+1]{k + S_{k-1}} = (k + S_{k-1})^{1/k+1} = pow(k + S_{k-1}, 1.0 / (k + 1))$

Bài 50: (trang 15)

```
#include <stdio.h>

int main() {
  int s, t, temp;

  printf( "Nhap s, t (0 < s < t): " );
  scanf( "%d%d", &s, &t );

  printf( "[" );
  do {
    printf( "%d", t / s );
    temp = t % s;
    /* tạo phân số t/s mới */
    t = s;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    s = temp;
    printf( temp ? ", " : "]\n" );
  } while ( temp );
  return 0;
}
```

Từ định nghĩa của phân số liên tục:

Ta thực hiện trong mỗi vòng lặp như sau:
- Tính thương số `b = t / s` và in ra.
- Hình thành nên phân số `t / s` mới, với tử số `= r = t % s` và mẫu số `= s`.

Chú ý các phép gán tạo phân số mới cần một biến tạm `temp` do `t` và `s` có liên quan với nhau.

Bài 51: (trang 16)

```
#include <stdio.h>

int main() {
  int i;
  float x, F;

  printf( "Nhap x: " );
  scanf( "%f", &x );

  F = x;
  for ( i = 256; i > 0; i /= 2 )
    F = x + i / F;
  printf( "F = %g\n", F );
  return 0;
}
```

Vấn đề của bài tập này là xác định vòng lặp cần thực hiện, xem biểu thức:

$$F = x + \cfrac{1}{x + \cfrac{2}{x + \cfrac{4}{\cdots \; x + \cfrac{128}{x + \cfrac{256}{x}}}}}$$

Dễ dàng nhận thấy công thức truy hồi của phân số liên tục trên:

$$F_k = x + \frac{2^k}{F_{k+1}}, k = 0 \div 9$$

Điều kiện đầu: $F_9 = x$

Đến đây ta có thể dùng vòng lặp `for`. Tuy nhiên, ta nhận thấy biểu thức được tính từ dưới lên trên, trong mỗi lần lặp ta phải chuyển 2^k thành $2^{k-1} = 2^k/2$. Do vậy ta dùng luôn $i = 2^k$ làm biến đếm.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
F = x;
for ( i = 256; i > 0; i /= 2 )
  F = x + i / F;
```

Cách viết đệ quy:

```
float F( float x, int n ) {
  if ( n > 256 ) return x;
  return x + n / F( x, n * 2 );
}

/* dùng trong hàm main() */
printf( "F = %g\n", F( x, 1 ) );
```

Bài 52: (trang 16)

```
#include <stdio.h>

int main() {
  int n, i;
  double s = 0.0;

  printf( "Nhap n: " );
  scanf( "%d", &n );

  for ( i = 1; i <= n; ++i )
    s += 1.0 / ( n * n + i );
  printf( "Fn = %g\n", s );
  return 0;
}
```

Dễ dàng nhận thấy cách diễn giải biểu thức toán học Σ với vòng lặp `for` có sự tương đồng: "tổng các $\frac{1}{n^2 + i}$, với `i` chạy từ `1` đến `n`"

$$\sum_{i=1}^{n} \frac{1}{n^2 + i}$$

```
for ( i = 1; i <= n; ++i )
  s += 1.0 / ( n * n + i );
```

Bài 53: (trang 16)

```
#include <stdio.h>
#include <math.h>
#define eps 1e-4

int main() {
  float s, expr, expo, fact, x;
  int i, sign = -1;

  printf( "Nhap x (radian): " );
  scanf( "%f", &x );

  fact = 1;
  s = expr = expo = x;
  for ( i = 3; expr > eps; i += 2, sign = -sign ) {
    expr = ( expo *= x * x ) / ( fact *= i * ( i - 1 ) );
    s += sign * expr;
  }
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  printf( "cong thuc Taylor: sin(%.2f) = %.4f\n", x, s );
  printf( "sin() cua math.h: sin(%.2f) = %.4f\n", x, sin( x ) );
  return 0;
}
```

Bài này tổng kết các kỹ năng dùng tính toán trong vòng lặp. Xét biểu thức:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

Viết gọn: $\sin(x) = \sum_{k=0} (-1)^k \frac{E_k}{F_k}$ với $E_k = x^{2k+1}, F_k = (2k+1)!$

Trong mỗi vòng lặp tử số E_k tăng: $\frac{x^{2k+1}}{x^{2(k-1)+1}} = \frac{x^2.x^{2k-1}}{x^{2k-1}} = x^2$

Trong mỗi vòng lặp mẫu số F_k tăng:

$$\frac{(2k+1)!}{(2(k-1)+1)!} = \frac{(2k+1).2k.(2k-1)!}{(2k-1)!} = (2k+1).2k$$

Suy ra biểu thức truy hồi: $\sin(x) = \sum_{k=0} (-1)^k \frac{E_k}{F_k} = \sum_{k=0} (-1)^k \frac{x^2 E_{k-1}}{(2k+1)2kF_{k-1}}$

Điều kiện đầu: $E_0 = x, F_0 = 1$

Giả sử không quan tâm đến việc dấu thay đổi tuần hoàn, đặt $i = 2k + 1$

$$\sum_{i=1} \frac{E_i}{F_i} = \sum_{i=1} \frac{x^2 E_{i-1}}{i(i-1)F_{i-1}} \quad (1)$$

Điều kiện đầu theo i: $E_1 = x, F_1 = 1$

Bây giờ ta có thể xây dựng vòng lặp for tính biểu thức expr với các chú ý sau:

- Biến đếm i được cập nhật i += 2 và khởi tạo i = 3 (do điều kiện đầu i = 1).
- Đổi dấu với từng vòng lặp, khởi tạo bằng trị âm (do điều kiện đầu dấu dương).
- Thân vòng lặp tính tích lũy luôn lũy thừa và giai thừa trong mỗi lần lặp:

```
expr = (biểu thức tính tích lũy lũy thừa) / (biểu thức tính tích lũy giai thừa);
```

Cụ thể, theo (1):

```
s += sign * ( expo *= x * x ) / ( fact *= i * ( i - 1 ) );
```

expo (tức E_i) khởi tạo bằng $E_1 = x$, fact (tức F_i) khởi tạo bằng $F_1 = 1$.

Tổng s, khởi tạo với dấu dương và trị: $\frac{E_1}{F_1} = x$.

Bài 54: (trang 16)

```
#include <stdio.h>

float combination( int k, int n ) {
  float i;
  float nfac = 1.0;
  for ( i = 1; i <= k; ++i )
    nfac *= ( ( n - i + 1 ) / i );
  return nfac;
}

int main() {
  int n, k;

  printf( "Nhap n, k (k < n < 25): " );
  scanf( "%d%d", &n, &k );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  printf( "C( k, n ): %g\n", combination( k, n ) );
  printf( "C( n - k, n ): %g\n", combination( n - k, n ) );
  return 0;
}
```

Cần biến đổi biểu thức yêu cầu thành dạng thuận lợi cho việc mô tả bằng cấu trúc lặp của ngôn ngữ lập trình:

$$C_n^k = \frac{n!}{k!\,(n-k)!} = \frac{n(n-1)(n-2)\ldots(n-k+2)(n-k+1)(n-k)!}{k!\,(n-k)!}$$

$$= \frac{n(n-1)(n-2)\ldots(n-k+2)(n-k+1)}{1.2\ldots k}$$

$$= \frac{n}{1}\cdot\frac{(n-1)}{2}\cdot\frac{(n-2)}{3}\cdots\frac{(n-(k-2))}{k-1}\cdot\frac{(n-(k-1))}{k}$$

$$= \prod_{i=1}^{k}\frac{n-i+1}{i}$$

dễ dàng mô tả công thức trên bằng phát biểu lặp.
Chú ý khai báo biến đếm `i` kiểu `float` để tránh phép chia số nguyên.

Bài 55: (trang 16)

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

double babylonian( int x ) {
  double y = 1.0, q;
  do {
    double p = x / y;
    y = ( y + p ) / 2;
    q = fabs( y - p );
  } while ( q > 1e - 13 );
  return y;
}

int main() {
  int x;

  printf( "Nhap x (x > 0): " );
  scanf( "%d", &x );

  printf( "thuat toan babylonian: %g\n", babylonian( x ) );
  printf( "ham sqrt() cua math.h: %g\n", sqrt( x ) );
  return 0;
}
```

Thực hiện thân vòng lặp chính xác theo từng bước được mô tả trong thuật toán Babylonian. Chú ý cách dùng hàm `fabs()` của thư viện `math.h` để tính trị tuyệt đối của một số thực là "khoảng cách" giữa `y` và `x / y`.

Bài 56: (trang 17)

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

void printBits( int n ) {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  int size = sizeof( int ) * CHAR_BIT;
  printf( "%d = ", n );
  while ( --size >= 0 ) {
    putchar( ( ( n >> size ) & 1 ) + '0' );
    if ( size % CHAR_BIT == 0 ) putchar( ' ' );
  }
  putchar( '\n' );
}

int main() {
  int n;

  printf( "Nhap n: " );
  scanf( "%d", &n );

  printBits( n );
  printf( "Hex: %X\n", n );
  return 0;
}
```

Hằng số `CHAR_BIT`, định nghĩa số bit trong `1` byte, được định nghĩa trước trong `limits.h`.

Ngôn ngữ C không có kích thước cố định cho từng kiểu dữ liệu, ngoại trừ kiểu `char` có kích thước `1 byte`. Kích thước kiểu `int` chẳng hạn, phụ thuộc vào kích thước thanh ghi và xác định được bằng toán tử `sizeof`.

Để xác định từng bit của một số `int`, theo thứ tự từ trái sang phải:

- Xác định số bit của biến `n` kiểu `int`: `size = sizeof( int ) * 8;`
- Dịch trái `n` một khoảng `(size - 1)` bit để đưa bit thứ nhất (kể từ bên trái) thành bit phải nhất.
- Thực hiện phép toán bitwise `AND` (toán tử `&` thao tác trên bit) giữa kết quả dịch phải ở trên với mặt nạ `1` để xác định bit phải nhất là `0` hoặc `1`.

Để xác định bit kế tiếp, thứ hai (kể từ bên trái), ta cũng đưa bit này thành bit phải nhất bằng cách dịch trái `n` một khoảng `(size - 1)` bit với `size` đã giảm xuống thêm `1` đơn vị.

Vì ta tính toán với `(size - 1)` nên giảm `size` với `--size`, rồi mới tính toán. Biến `size` cũng dùng làm biến đếm dùng tách biệt các cụm bit của mỗi byte để dễ xem.

Dạng hiển thị thập lục phân (hexadecimal, hệ đếm 16) hoặc bát phân (octal, hệ đếm 8) đã được ngôn ngữ C hỗ trợ:

```
printf( "%d(d) = %#x(h) = %#o(o)\n", 1234, 1234, 1234 );
printf( "%#x(h) = %d(d)\n", 0x162E, 0x162E );
printf( "%#o(o) = %d(d)\n", 013056, 013056 );
```

Bài 57: (trang 17)

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

unsigned parity( int n ) {
  int c, size = sizeof( int ) * CHAR_BIT;
  unsigned b = 0;
  for ( c = size - 1; c >= 0; --c )
    b += ( n >> c ) & 1;
  return ( b % 2 );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
}

int main() {
  int n;

  printf( "Nhap n: " );
  scanf( "%d", &n );

  printf( "Even parity bit = %u\n", parity( n ) );
  return 0;
}
```

Một cách đếm các bit 1 khác với bài 56 (trang 111): dịch trái n sao cho các bit của n tính từ bit MSB sang phải, lần lượt được đưa đến vị trí bit 0. Dùng toán tử bitwise AND với mặt nạ 1, kiểm tra bit tại vị trí bit 0 và đếm các bit 1.
Chú ý cách trả về của hàm parity(), nếu cần tìm bit parity lẻ, đơn giản thay đổi trị trả về như sau: return !(b % 2);
Một cách tính bit parity chẵn hay, nên tham khảo:

```
unsigned parity( int n ) {
  unsigned b;
  for ( b = 0; n != 0; n &= ( n - 1 ) ) b++;
  return ( b % 2 );
}
```

Bài 58: (trang 17)

```
#include <stdio.h>
#define MAX 100

int main() {
  int i, j, n;
  int a[MAX] = { 0 };

  printf( "Nhap n: " );
  scanf( "%d", &n );

  for ( i = 2; i <= n; ++i )
    if ( !a[i] ) {
      printf( "%d ", i );
      /* vòng lặp "gạch chéo" các bội số của i */
      for ( j = i + i; j <= n; j += i ) a[j] = 1;
    }
  putchar( '\n' );
  return 0;
}
```

Ta chỉ dùng một mảng a, trong đó:
- Chỉ số của mảng là dãy số cần "sàng" để xác định các số nguyên tố.
- Trị của từng phần tử: nếu bằng 1, số trong dãy số (tương ứng với chỉ số của phần tử) sẽ bị "gạch chéo".

Khi duyệt mảng a, bất cứ chỉ số nào có trị bằng 0 ta gặp đều là số nguyên tố, vì nếu không là số nguyên tố thì trị tại chỉ số đó đã được chuyển thành 1 do quá trình "gạch chéo" bội số của một ước số nguyên tố nào đó *của nó* trước đó (chú ý chỉ số đầu tiên 2 chắc chắn là số nguyên tố).

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Vòng lặp "gạch chéo" bội số một chỉ số (chuyển trị của chỉ số thành `1`) của số nguyên tố tìm thấy, được thực hiện bằng phép cộng nên rất nhanh, vì bước lặp tăng dần nên số bước lặp càng ít lại.
Cách cài đặt trên sát với thuật toán, ngắn gọn và hiệu quả.
Chú ý cách khởi tạo mảng `a` chứa toàn `0` bằng khai báo:
`int a[MAX] = { 0 };`
Cũng có thể khai báo mảng `a` là `static` để ngay từ đầu các phần tử của mảng đã được khởi tạo trị `0`, nhưng luôn cẩn thận khi sử dụng biến kiểu `static`.

Bài 59: (trang 18)

```
#include <stdio.h>

int main() {
  int n;
  char can[][5] = {"Canh", "Tan", "Nham", "Quy", "Giap",
                   "At", "Binh", "Dinh", "Mau", "Ky"};
  char chi[][5] = {"Than", "Dau", "Tuat", "Hoi", "Ti", "Suu",
                   "Dan", "Meo", "Thin", "Ty", "Ngo", "Mui"};
  printf( "Nhap nam: " );
  scanf( "%d", &n );

  printf( "%d - %s %s\n", n, can[ n%10 ], chi[ n%12 ] );
  printf( "%d - %s %s\n", n + 60, can[ n%10 ], chi[ n%12 ] );
  return 0;
}
```

Mặc dù Can bắt đầu từ Giáp và Chi bắt đầu từ Tí, nhưng do năm `0` làm mốc là năm Canh Thân, nên các mảng `can` và `chi` được bố trí như trên cho phù hợp.
Chú ý cách dùng toán tử `%`, hạn chế chỉ số của mảng `can` (trong `[0, 10)`) và mảng `chi` (trong `[0, 12)`).
Phương pháp dùng bảng tra cứu (lookup table) như trên được sử dụng rất phổ biến.

Bài 60: (trang 18)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define MAX 100

void shuffle( int a[], int n ) {
  int t[MAX], i;
  for ( i = 0; i < n / 2; ++i ) {
    t[2 * i] = a[i];
    t[2 * i + 1] = a[n / 2 + i];
  }
  memmove( a, t, ( n - 1 ) * sizeof( *a ) );
}

int isEqual( int a[], int b[], int n ) {
  int i;
  for ( i = 0; i < n; ++i )
    if ( a[i] != b[i] ) return 0;
  return 1;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
}

int main() {
  int a[MAX], b[MAX];
  int n, i, count;

  srand( time( NULL ) );
  do {
    printf( "Nhap n (n chan): " );
    scanf( "%d", &n );
  } while ( n < 2 || n % 2 );

  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] = b[i] = rand() % 201 - 100 );
  putchar( '\n' );

  shuffle( a, n );
  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] );

  shuffle( b, n );
  count = 0;
  do {
    shuffle( b, n );
    count++;
  } while ( !isEqual( a, b, n ) );
  printf( "\nCan %d lan shuffle de mang tro ve ban dau\n", count );
  return 0;
}
```

Đoạn `[a, b]` chứa `(b - a + 1)` số. Xem bài 11 (trang 73), để sinh `n` số ngẫu nhiên trong nửa đoạn `[0, n)`, dùng `rand() % n`. Suy ra, để sinh `(b - a + 1)` số ngẫu nhiên trong nửa đoạn `[0, b - a + 1)`, dùng `rand() % (b - a + 1)`.
Vậy để sinh số ngẫu nhiên trong đoạn `[a, b]`, tương đương nửa đoạn `[a, b + 1)`, ta điều chỉnh cận trên và cận dưới, dùng `rand() % (b - a + 1) + a`.
Bài tập: Xuất số ngẫu nhiên trong tập các số lẻ từ 3 đến 99.
Số lẻ trong đoạn `[3, 99]` có dạng `2 * k + 1` với `k ∈ [1, 49]`, bài tập trở thành: tìm số ngẫu nhiên `k ∈ [1, 49]`.

```
int randomOdd() {
  return 2 * ( rand() % 49 + 1 ) + 1;
}
```

Hình bên minh họa một lần trộn perfect shuffle mảng `a` thành mảng tạm `t`, được thực hiện trong hàm `shuffle()`, chú ý cách tính các chỉ số với những mục đích khác nhau.

Vòng lặp trộn chỉ cần *chạy đến nửa mảng* vì trong thân vòng lặp thực hiện đến hai thao tác chuyển vị. Sau khi trộn xong mảng `t` sẽ được sao chép trở lại mảng gốc `a`. Bạn có thể thực hiện nhanh thao tác này bằng cách chuyển khối vùng nhớ cấp cho mảng `t` chồng lên khối vùng nhớ cấp cho mảng `a`, bằng hàm chuẩn `memmove()` của thư viện `<string.h>`.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Hàm `memmove()` có 3 tham số: địa chỉ đầu vùng nhớ đích, địa chỉ đầu vùng nhớ nguồn (tham số nằm bên phải tham số địa chỉ vùng nhớ đích, bố trí giống như phép gán) và kích thước tính bằng byte của khối vùng nhớ cần chuyển.
Để chấm dứt vòng lặp đếm số lần *perfect shuffle* cho mảng trở về ban đầu, ta kiểm tra kết quả trộn bằng hàm so sánh hai mảng `isEqual()`.

Bài 61: (trang 18)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 100

int main() {
  int a[MAX];
  int n, i, s, p;

  srand( time( NULL ) );
  do {
    printf( "Nhap n [1, %d]: ", MAX - 1 );
    scanf( "%d", &n );
  } while ( n < 1 || n > MAX - 1 );

  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] = rand() % 201 - 100 );
  for ( s = i = 0; i < n; ++i )
    if ( a[i] > 0 ) s += a[i];
  printf( "\nTong cac so nguyen duong = %d\n", s );

  do {
    printf( "Nhap p [0, %d]: ", n - 1 );
    scanf( "%d", &p );
  } while ( p < 0 || p > n - 1 );
  for ( i = p + 1; i < n; ++i )
    a[i-1] = a[i];
  n--;

  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] );
  putchar( '\n' );
  return 0;
}
```

Thông thường, việc xóa mảng tại một vị trí `p` được thực hiện bằng cách dịch chuyển dồn từng phần tử của mảng sau vị trí `p` chồng lên phần tử ngay trước nó. C không kiểm tra việc truy xuất vượt quá phạm vi của mảng, lập trình viên phải chịu trách nhiệm kiểm soát kích thước của mảng, vì vậy phải nhớ giảm số phần tử của mảng xuống sau khi xóa phần tử.

C lưu trữ các phần tử của một mảng trong một vùng nhớ vật lý liên tục, vì vậy để dịch chuyển các phần tử nhanh hơn, người ta thường dùng hàm chuẩn `memmove()` của `string.h`:

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

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
/* ... */
memmove( &a[p], &a[p + 1], ( n - p - 1 ) * sizeof( *a ) );
```

ta đã chuyển khối vùng nhớ có (`n - p - 1`) phần tử, bắt đầu từ `a[p + 1]` chồng lên khối vùng nhớ bắt đầu từ `a[p]`.

Chú ý trong C: `a = &a[0]` là địa chỉ bắt đầu của mảng `a`; `a + p = &a[p]` là địa chỉ của phần tử `a[p]`. Điều này rất quen thuộc với lập trình viên C (gọi là "idiom của C"), vì vậy bạn nên viết như sau:

```
memmove( a + p, a + p + 1, ( n - p - 1 ) * sizeof( *a ) );
```

Lập trình viên C xem `a[2]` chẳng hạn, như một *địa chỉ*, nên viết `a[2]` hoặc `2[a]` cũng cho kết quả như nhau vì: `a[i]` tương đương `*(a + i)`, tương đương `*(i + a)` và tương đương `i[a]`. Tất nhiên, dùng `a[i]` thì quen thuộc hơn.

Bài 62: (trang 19)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 100

int gcd( int a, int b ) {
  return ( !a ) ? b : gcd( b % a, a );
}

int main() {
  int a[MAX], n, i, j, s1, s2;

  srand( time( NULL ) );
  do {
    printf( "Nhap n [1, %d]: ", MAX - 1 );
    scanf( "%d", &n );
  } while ( n < 1 || n > MAX - 1 );
  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] = rand() % 11 + 10 );

  for ( s1 = s2 = i = 0; i < n; ++i ) {
    if ( i % 2 && a[i] % 2 == 0  ) s1 += a[i];
    if ( i % 2 == 0 && a[i] % 2  ) s2 += a[i];
  }
  printf( "\nTong le vi tri chan (%d) ", s1 );
  printf( s1 == s2 ? "bang" : "khac" );
  printf( " tong chan vi tri le (%d)\n", s2 );

  printf( "Cac cap nguyen to cung nhau:\n" );
  for ( i = 0; i < n; ++i )
    for ( j = i + 1; j < n; ++j )
      if ( gcd( a[i], a[j] ) == 1 )
        printf( "(%2d, %2d)\n", a[i], a[j] );
  return 0;
}
```

Khi xét các cặp nguyên tố cùng nhau, vì quan hệ này là quan hệ không thứ tự nên có thể gặp các cặp kết quả trùng lặp, ví dụ: `(a[5], a[8])` với `(a[8], a[5])`. Do đó, ta chỉ xét một phần tử với dãy phần từ ngay sau nó đến cuối mảng. Dễ dàng nhận thấy

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

các cặp kết quả luôn có chỉ số phần tử sau lớn hơn chỉ số phần tử trước, tránh được việc xuất các cặp kết quả trùng lặp.

Tuy nhiên ta vẫn nhận được các kết quả trùng lặp khi chạy chương trình. Điều này do miền giá trị hẹp dẫn đến sự trùng lặp trị của các phần tử. Ví dụ, giả sử `a[10]` có trị trùng `a[8]`, ta có cặp kết quả trùng lặp: `(a[5], a[8])` với `(a[5], a[10])`. Để giải quyết triệt để, trước tiên cần xóa các phần tử trùng nhau trong mảng, chỉ chừa lại một phần tử. Bạn tham khảo thêm bài 64 (trang 119) và bài 65 (trang 120), ví dụ có thể thực hiện như sau:

```
for ( i = 0; i < n; ++i )
  for ( j = i + 1; j < n; ++j )
    if ( a[j] == a[i] ) a[j--] = a[--n];
for ( i = 0; i < n; ++i )
  for ( j = i + 1; j < n; ++j )
    if ( gcd( a[i], a[j] ) == 1 )
      printf( "(%2d, %2d)\n", a[i], a[j] );
```

Bài 63: (trang 19)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#define MAX 100

int main() {
  int a[MAX];
  int n, i, c;

  srand( time( NULL ) );
  do {
    printf( "Nhap n [1, %d]: ", MAX - 1 );
    scanf( "%d", &n );
  } while ( n < 1 || n > MAX - 1 );
  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] = rand() % 201 - 100 );

  for ( c = i = 0; i < n; ++i )
    if ( a[i] % 4 == 0 && abs( a[i] ) % 10 == 6 ) c++;
  printf( "\nCo %d phan tu chia het cho 4, tan cung 6\n", c );

  printf( "Nhan doi phan tu le:\n" );
  for ( i = 0; i < n; ++i )
    if ( a[i] % 2 ) a[i] *= 2;

  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] );
  putchar( '\n' );
  return 0;
}
```

Điều kiện một số tận cùng là 6 không khó, ta tách chữ số cuối bằng cách lấy phần dư của phép chia cho 10: $\overline{ab} = 10a + b$. Tuy nhiên, bạn dễ bỏ sót trường hợp `a[i]` tận cùng là 6 nhưng có trị âm nên `a[i] % 10` bằng `-6`, không phải `6`.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 64: (trang 19)

```
#include <stdio.h>
#define MAX 100
#define ispow2(x) (!((x) & ((x)-1)) && x)

int main() {
  int a[MAX];
  int n, i, c, x;

  do {
    printf( "Nhap n [1, %d]: ", MAX - 1 );
    scanf( "%d", &n );
  } while ( n < 1 || n > MAX - 1 );

  printf( "Nhap %d phan tu:\n", n );
  for ( i = 0; i < n; ++i )
    scanf( "%d", a + i );

  for ( c = i = 0; i < n; ++i )
    if ( ispow2( a[i] ) ) c++;
  printf( "Co %d so la luy thua cua 2\n", c );

  printf( "Nhap x: " );
  scanf( "%d", &x );
  for ( c = i = 0; i < n; ++i )
    if ( a[i] != x ) a[c++] = a[i];
  n = c;

  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] );
  putchar( '\n' );
  return 0;
}
```

Macro `ispow2(x)` dùng kiểm tra `x` có phải là lũy thừa của 2 không, đã được giải thích trong bài 48 (trang 105).

Để xóa phần tử có trị bằng `x` trong mảng `a` có `n` phần tử, theo bài 61 (trang 116):

```
for ( i = 0; i < n; ++i )
  if ( a[i] == x ) {
    memmove( a + i, a + i + 1, (n - i - 1 ) * sizeof( *a ) );
    i--;
    n--;
  }
```

Sau khi dồn mảng, do `a[i]` bị phần tử sau nó chồng lên, ta cần lùi `i` (`i--`) để *xét lại phần tử* `a[i]`. Có thể dùng một phương pháp hay: duyệt ngược mảng, như vậy tránh được tình trạng phần tử chưa kịp xét chồng lên phần tử vừa bị xóa:

```
for ( i = n - 1; i >= 0; --i )
  if ( a[i] == x ) {
    memmove( a + i, a + i + 1, (n - i - 1 ) * sizeof( *a ) );
    n--;
  }
```

Ta vẫn có thể xóa một phần tử trong mảng mà không cần dồn mảng: lấy phần tử cuối chồng lên phần tử cần xóa; đồng thời vừa lùi `i` (tránh trường hợp phần tử cuối mảng bằng `x`), vừa trừ số phần tử của mảng `1` đơn vị:

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
for ( i = 0; i < n; ++i )
  if ( a[i] == x )
    a[i--] = a[--n];
```

Phương pháp này chỉ thực hiện được trên mảng không thứ tự, những mảng có thứ tự, ví dụ như chuỗi, không thể áp dụng cách này. Cách này hiệu quả trong việc xóa một phần tử bất kỳ trong danh sách liên kết (không có thứ tự).

Trong bài tập này và một số bài tập sau, chúng tôi dùng một phương pháp khác đặc biệt dễ hiểu: để loại các phần tử có trị bằng `x` ta sao chép các phần tử có trị khác `x` sang một mảng `b` khác:

```
for ( k = i = 0; i < n; ++i )
  if ( a[i] != x )
    b[k++] = a[i];
/* k là số phần tử của mảng b */
```

Như vậy ta phải khai báo thêm mảng `b`. Tuy nhiên, nhận thấy rằng lúc nào `k ≤ i`, ta *dùng lại mảng* `a` mà *không cần khai báo thêm mảng* `b`:

```
for ( k = i = 0; i < n; ++i )
  if ( a[i] != 2 )
    a[k++] = a[i];
n = k;       /* n là số phần tử mới của mảng a */
```

Có thể viết tối ưu hơn với nhận xét: không cần sao chép từ `a[i]` đến `a[k]` nếu `k = i` (nghĩa là không cần sao chép một phần tử lên *chính nó*):

```
for ( k = i = 0; i < n; ++i )
  if ( a[i] != x ) {
    if ( k != i ) a[k] = a[i];
    k++;
  }
n = k;
```

Bài 65: (trang 20)

```
#include <stdio.h>
#define MAX 100

int main() {
  int a[MAX], n;
  int i, j, c, s;

  do {
    printf( "Nhap n [1, %d]: ", MAX - 1 );
    scanf( "%d", &n );
  } while ( n < 1 || n > MAX - 1 );

  printf( "Nhap %d phan tu:\n", n );
  for ( i = 0; i < n; ++i )
    scanf( "%d", a + i );

  for ( s = c = i = 0; i < n; ++i )
    if ( a[i] < 0 && a[i] % 2 ) {
      c++;
      s += a[i];
    }
  printf( "Trung binh cong nguyen am le = %.2f\n", c ? ( float ) s / c : 0 );
  for ( i = 0; i < n - 1; ++i ) {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    for ( c = j = i + 1; j < n; ++j )
      if ( a[j] != a[i] ) a[c++] = a[j];
    n = c;
  }

  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] );
  putchar( '\n' );
  return 0;
}
```

Tính trung bình các phần tử nguyên âm của một mảng có vẻ đơn giản. Tuy nhiên bạn có thể gặp vài lỗi:

- Nếu không xét trường hợp mảng không có phần tử nguyên âm nào, do số phần tử nguyên âm `c` đếm được bằng `0`, bạn sẽ gặp lỗi chia cho `0`.
- Phép chia để lấy trung bình cộng là phép chia nguyên. Để giải quyết, ta cần một trong các toán hạng của biểu thức là số thực:

bằng cách ép kiểu: `( float ) s / c`
bằng cách khai báo trước `s` có kiểu `float`
hoặc bằng cách thêm một toán hạng là số thực: `s * 1.0 / c`

Khi định trị một biểu thức với các toán hạng có nhiều kiểu khác nhau, C sẽ nâng cấp kiểu các toán hạng một cách không tường minh rồi mới thực hiện định trị.

Cách xóa tất cả phần tử có trị `x` trong mảng `a`, trình bày trong bài 64 (trang 119) thuận lợi cho việc viết các chương trình xóa phần tử trong mảng với điều kiện phức tạp như bài tập trên:

```
for ( i = 0; i < n - 1; ++i ) {
  for ( c = j = i + 1; j < n; ++j )
    if ( a[j] != a[i] ) a[c++] = a[j];
  n = c;
}
```

Với phần tử `a[i]` đang xét ta xóa tất cả phần tử có trị trùng `a[i]` trong mảng con ngay sau nó (phần mảng từ `j = i + 1` trở đi). Chỉ xét đến phần tử `n - 2` vì phần tử cuối chắc chắn không còn phần tử trùng (không có mảng con phía sau).

Vì `&a[i]` tương đương `a + i`, nên thay vì viết: `scanf( "%d", &a[i] );`
ta có thể viết tắt: `scanf( "%d", a + i );`

Bài 66: (trang 20)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 100

int main() {
  int a[MAX];
  int n, i, k, maxpos, minpos;

  srand( time( NULL ) );
  do {
    printf( "Nhap n [1, %d]: ", MAX - 1 );
    scanf( "%d", &n );
  } while ( n < 1 || n > MAX - 1 );
  for ( i = 0; i < n; ++i )
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    printf( "%d ", a[i] = rand() % 201 - 100 );
  putchar( '\n' );

  maxpos = minpos = 0;
  for ( i = 1; i < n; ++i ) {
    if ( a[i] < a[minpos] ) minpos = i;
    if ( a[i] > a[maxpos] ) maxpos = i;
  }
  printf( "max = %d\n", a[maxpos] );
  printf( "min = %d\n", a[minpos] );

  for ( k = i = maxpos + 1; i < n; ++i )
    if ( a[i] != a[maxpos] ) a[k++] = a[i];
  n = k;

  for ( i = 0; i < n; ++i )
    printf( "%d ",a[i] );
  putchar( '\n' );
  return 0;
}
```

Có thể tìm trị lớn nhất như sau:

```
maxval = a[0];
for ( i = 1; i < n; ++i )
  if ( a[i] > maxval ) maxval = a[i];
printf( "max = %d\n", maxval );
```

Ta giả sử phần tử đầu tiên của mảng là phần tử lớn nhất, gọi là `maxval`. Tiến hành so sánh trị `maxval` này với các phần tử còn lại trong mảng, nếu xuất hiện phần tử nào lớn hơn `maxval` thì hiệu chỉnh lại giá trị `maxval`.

Giải thuật tìm trị nhỏ nhất của mảng thực hiện tương tự.

Tuy nhiên, khi làm việc với mảng bạn nên tập *tư duy theo chỉ số*. Ví dụ ta tiến hành tìm trị lớn nhất trong một mảng như sau:

```
maxpos = 0;
for ( i = 1; i < n; ++i )
  if ( a[i] > a[maxpos] ) maxpos = i;
printf( "max = %d\n", a[maxpos] );
```

Ta giả sử phần tử đầu tiên (chỉ số `0`) của mảng là phần tử lớn nhất, ta lưu chỉ số thay vì lưu phần tử: `maxpos = 0`. Tiến hành duyệt mảng từ phần tử thứ hai (chỉ số `1`) trở đi, nếu phần tử đang xét lớn hơn phần tử có chỉ số `maxpos` đã lưu, ta cập nhật lại chỉ số `maxpos` với chỉ số đang xét.

Bài 67: (trang 20)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 100

int main() {
  int a[MAX], t;
  int n, i, j;

  srand( time( NULL ) );
  do {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    printf( "Nhap n [1, %d]: ", MAX - 1 );
    scanf( "%d", &n );
  } while ( n < 1 || n > MAX - 1 );
  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] = rand() % 201 - 100 );
  putchar( '\n' );

  for ( i = 0; i < n - 1; ++i )
    for ( j = i + 1; j < n; ++j )
      if ( ( a[i] % 2 == 0 && a[j] % 2 == 0 && a[i] > a[j] ) ||
           ( a[i] % 2 && a[j] %2 && a[i] < a[j] ) )
        { t = a[i]; a[i] = a[j]; a[j] = t; }

  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] );
  putchar( '\n' );
  return 0;
}
```

Bài tập này yêu cầu nắm vững giải thuật sắp xếp (sort). Các giải thuật sắp xếp được trình bày trong các giáo trình Cấu trúc dữ liệu và giải thuật hoặc Kỹ thuật lập trình, mỗi giải thuật được thực hiện bằng nhiều cách khác nhau. Trong tập sách này, để bạn dễ hiểu và dễ nhớ giải thuật, chúng tôi chỉ dùng giải thuật đơn giản, không sát với giải thuật gốc.

Giả sử ta cần sắp xếp tăng `n` phần tử của mảng `a`. Giải thuật sắp xếp thường được sử dụng nhất trong các bài tập lập trình cơ bản là sắp xếp theo kiểu chọn (Selection Sort). Nội dung như sau:

```
for ( i = 0; i < n - 1; ++i )
  for ( j = i + 1; j < n; ++j )
    if ( a[i] > a[j] )
      /* hoán chuyển a[i] với a[j] */
      t = a[i]; a[i] = a[j]; a[j] = t;
```

Vòng lặp ngoài, chỉ số `i`, thực hiện `n - 1` đợt sắp xếp, mỗi đợt sắp xếp đúng cho một vị trí `i`. Mảng được sắp xếp theo thứ tự từ đầu đến cuối.

Để sắp xếp đúng một vị trí `i`, ta so sánh `a[i]` với các phần tử sau nó (từ `j = i + 1` trở đi), chọn ra phần tử nhỏ nhất đặt vào vị trí `a[i]`. Vòng lặp trong, chỉ số `j`, thực hiện điều này: duyệt các phần tử sau vị trí `i`, phần tử đang duyệt `a[j]` nào nhỏ hơn phần tử `a[i]` đang xét thì hoán đổi ngay với `a[i]`.

Nhắc lại, sắp xếp (tăng) theo kiểu chọn như trên chỉ dễ nhớ, nhưng chưa thể hiện rõ ý tưởng giải thuật gốc, có số lần hoán đổi ít hơn: trong phân đoạn mảng chưa sắp xếp, hoán đổi phần tử đầu với phần tử nhỏ nhất.

Từ cách chọn phần tử lớn nhất hay nhỏ nhất của một mảng trong bài 66 (trang 121) bạn dễ dàng thực hiện điều này:

```
for ( i = 0; i < n - 1; ++i ) {
  /* Duyệt phân đoạn mảng chưa sắp xếp */
  for ( minpos = i, j = i + 1; j < n; ++j )
    /* Chọn lại minpos nếu phát hiện phần tử có trị nhỏ hơn */
    if ( a[j] < a[minpos] ) minpos = j;
      /* Hoán chuyển phần tử đầu với phần tử nhỏ nhất tại vị trí minpos */
      t = a[i], a[i] = a[minpos]; a[minpos] = t;
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Việc hoán chuyển hai phần tử với nhau có thể dùng macro hay gọi hàm (truyền tham số bằng con trỏ).
Tư duy theo chỉ số như trên giúp bạn giải một số bài tập phức tạp, ví dụ bài tập sau.
Bài tập: In các phần tử của một mảng cho trước theo thứ tự tăng dần, mà vẫn giữ nguyên vị trí của các phần tử trong mảng đó.

```
Mang ban dau  : 8 3 6 7 2 5 7 1 4 1
Sap xep tang  : 1 1 2 3 4 5 6 7 7 8
Mang sau xu ly: 8 3 6 7 2 5 7 1 4 1
```

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

int* sortAscIndex( int* a, int n ) {
  int *p = malloc( n * sizeof( int ) );
  int i, j, minpos;
  for ( i = 0; i < n; ++i ) p[i] = i;
  for ( i = 0; i < n - 1; ++i ) {
    int t;
    for ( minpos = i, j = i + 1; j < n; ++j )
      if ( a[p[j]] < a[p[minpos]] ) minpos = j;
    t = p[i]; p[i] = p[minpos]; p[minpos] = t;
  }
  return p;
}

int main() {
  int a[] = { 8, 3, 6, 7, 2, 5, 7, 1, 4, 1 };
  int *p;
  int i, size = sizeof a / sizeof *a;

  printf( "Mang ban dau  : " );
  for ( i = 0; i < size; ++i ) printf( "%d ", a[i] );
  putchar( '\n' );
  p = sortAscIndex( a, size );
  printf( "Sap xep tang  : " );
  for ( i = 0; i < size; ++i ) printf( "%d ", a[p[i]] );
  putchar( '\n' );
  printf( "Mang sau xu ly: " );
  for ( i = 0; i < size; ++i ) printf( "%d ", a[i] );
  putchar( '\n' );
  free( p );
  return 0;
}
```

Ta tạo một *mảng chỉ số* ánh xạ chỉ số của mảng cần sắp xếp. Khi so sánh các phần tử và thấy cần sắp xếp, thay vì sắp xếp lại vị trí của các phần tử trong mảng gốc, ta sắp xếp lại vị trí của các *chỉ số* trong mảng *chỉ số*.
Như vậy, mảng chỉ số kết quả lưu các chỉ số của mảng gốc, theo thứ tự sao cho các phần tử của mảng gốc tại các chỉ số đó có thứ tự tăng dần.

Bài 68: (trang 20)

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

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
#include <math.h>
#define MAX 100
#define swap(a, b) { int t = a; a = b; b = t; }

int main() {
  int a[MAX], n, i, j, i1, j1, k;

  srand( time( NULL ) );
  do {
    printf( "Nhap n (n chan): " );
    scanf( "%d", &n );
  } while ( n < 2 || n % 2 );
  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] = rand() % 101 + 100 );
  putchar( '\n' );

  for ( k = 0; k < n/2; ++k ) {
    /* Tìm cặp (a[i1], a[j1]) sao cho |a[i1] - a[j1]| nhỏ nhất
       đoạn cần xét từ từ thu hẹp lại:
       0 → n-1, 1 → n-2, ... , k → n-k-1, ... */
    int min = 101;
    for ( i = k; i < n - k; ++i )
      for ( j = i + 1; j < n - k; ++j )
        if ( abs( a[i] - a[j] ) < min ) {
          min = abs( a[i] - a[j] );
          i1 = i;
          j1 = j;
        }
    /* Nếu hiệu a[i1] - a[j1] < 0, hoán chuyển chúng */
    if ( a[i1] < a[j1] ) swap( a[i1], a[j1] );
    /* a[i1] chuyển về đầu và a[j1] chuyển về cuối đoạn đang xét */
    swap( a[i1], a[k] );
    swap( a[j1], a[n-k-1] );
  }
  for ( j = i = 0; i < n/2; ++i ) { /* In và tính tổng nửa mảng đầu */
    printf( "%d ", a[i] );
    j += a[i];
  }
  printf( ": %d\n", j );
  for ( j1 = 0; i < n; ++i ) {     /* In tiếp và tính tổng nửa mảng sau */
    printf( "%d ", a[i] );
    j1 += a[i];
  }
  printf( ": %d\n", j1 );
  printf( "Hieu nho nhat = %d\n", j - j1 );
  return 0;
}
```

Kết hợp các kỹ thuật của bài 62 (trang 117: tìm cặp phần tử thỏa yêu cầu cho trước) và bài 66 (trang 121: xác định max hoặc min của một dãy), ta tìm được cặp a_0, b_0 ($a_0 > b_0$) có hiệu nhỏ nhất, cặp a_1, b_1 ($a_1 > b_1$) có hiệu nhỏ thứ hai, …

Hoán chuyển trong mảng để chúng được sắp xếp như sau: a_0, a_1, …, b_1, b_0. Việc hoán chuyển này vừa tổ chức lại mảng tiện cho xuất kết quả, vừa giúp tìm các cặp kế tiếp được dễ dàng, vì sau mỗi lần tìm không gian tìm kiếm lại thu hẹp 2 phần tử.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Như vậy hiệu ($a_0 + a_1 + \ldots$) - ($b_0 + b_1 + \ldots$) = tổng nửa mảng đầu - tổng nửa mảng sau sẽ nhỏ nhất.

Trong trường hợp hai phần tử có trị khác biệt nhau nhất: a_0 bằng biên phải và a_1 bằng biên trái của đoạn `[100, 200]`, hiệu của chúng (`100`) vẫn nhỏ hơn trị `min` được gán đầu tiên (`101`). Nghĩa là lúc nào cũng tìm được cặp phần tử thỏa yêu cầu.

Bài 69: (trang 21)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 100

int main() {
  int a[MAX];
  int i, head, n, maxlen, maxhead;

  srand( time( NULL ) );
  do {
    printf( "Nhap n [1, %d]: ", MAX - 1 );
    scanf( "%d", &n );
  } while ( n < 1 || n > MAX - 1 );

  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] = rand() % 201 - 100 );
  putchar( '\n' );

  head = maxhead = maxlen = 0;
  do {
    int len = 1;
    for ( i = head; i < n - 1 && a[i] < a[i+1]; ++i )
      len++;
    if ( len > maxlen )
      { maxlen = len; maxhead = head; }
    head = i + 1;
  } while ( head < n );

  printf( "\"run\" tang dai nhat: " );
  for ( i = 0; i < maxlen; ++i )
    printf( "%d ", a[maxhead + i] );
  putchar( '\n' );
  return 0;
}
```

Một "run" trong quá trình dò tìm được xác định bằng 2 biến:

- `head`: chỉ số phần tử đầu của "run", được cập nhật là phần tử ngay sau "run" trước, khi dò tìm "run" mới. Ban đầu `head` được khởi tạo bằng `0`.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

- `len`: chiều dài của "run", là số phần tử đếm được khi phần tử đang xét vẫn còn thỏa tính chất của "run".

Để so sánh giữa các "run", dùng 2 biến lưu kết quả dò tìm:

- `maxlen`: lưu chiều dài "run" dài nhất hiện tại, sẽ được cập nhật nếu phát hiện "run" có chiều dài lớn hơn.
- `maxhead`: lưu chỉ số phần tử `head` của "run" dài nhất hiện tại, nghĩa là có chiều dài tương ứng `maxlen`.

Bài 70: (trang 21)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 100

void Swap( int *a, int *b ) {
  int t = *a; *a = *b; *b = t;
}

int main() {
  int a[MAX];
  int n, i, j;

  srand( time( NULL ) );
  do {
    printf( "Nhap n [1, %d]: ", MAX - 1 );
    scanf( "%d", &n );
  } while ( n < 1 || n > MAX - 1 );

  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] = rand() % 201 - 100 );
  putchar( '\n' );

  /* j chỉ phần tử đầu mảng, i dùng để duyệt
     nếu a[i] lẻ, hoán chuyển a[i] với a[j] và tăng j */
  for ( i = j = 0; i < n; ++i )
    if ( a[i] % 2 ) Swap( a + i, a + j++ );
  /* số lẻ cuối cùng là a[j - 1],
     j chỉ phần tử đầu mảng còn lại, i dùng để duyệt tiếp
     nếu a[i] bằng 0, hoán chuyển a[i] với a[j] và tăng j */
  for ( i = j; i < n; ++i )
    if ( !a[i] ) Swap( a + i, a + j++ );

  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] );
  putchar( '\n' );
  return 0;
}
```

Tưởng chừng bài tập có thể thực hiện dễ dàng như sau:

```
do {
  while ( a[i] % 2 && i < j ) i++;
  while ( a[j] % 2 == 0 && j > i ) j--;
  if ( i < j ) swap( a[i], a[j] );
} while ( i < j );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

vòng lặp `while` với biến đếm `i` dùng theo dõi "run" lẻ (nếu có) từ đầu mảng trở đi, vòng lặp `while` với biến đếm `j` dùng theo dõi "run" chẵn (nếu có) từ cuối mảng về trước. Khi các "run" này chấm dứt, các trường hợp sau đây xảy ra:

- Nếu `i < j`, nghĩa là hai "run" đều bị dừng do xuất hiện phần tử khác tính chất của mỗi "run". Ta hoán chuyển hai phần tử này với nhau rồi tiếp tục duyệt tiếp hai "run" (vòng lặp `do while` bên ngoài).
- Ngược lại, hai "run" đã gặp nhau, hoặc một trong hai "run" đã duyệt hết mảng (mảng toàn chẵn hoặc toàn lẻ), ta kết thúc vòng lặp duyệt bên ngoài.

Tuy nhiên, cách này không giải quyết được các phần tử có trị `0`. Không có cách xử lý dù phát hiện được chúng trong "run", ví dụ trong trường hợp hai "run" đều dừng khi phát hiện có trị `0`.

Bài 71: (trang 21)

```
#include <stdio.h>
#include <string.h>
#define MAX 100

void lshiftkstep( int a[], int n, int k ) {
  int i;
  k %= n;
  for ( i = 0; i < k; ++i ) {
    int t = a[0];
    memmove( a, a + 1, ( n - 1 ) * sizeof( *a ) );
    a[n - 1] = t;
  }
}

int isSymmetrical( int a[], int n ) {
  int i;
  for ( i = 0; i < n / 2; ++i )
    if ( a[i] != a[n - 1 - i] ) return 0;
  return 1;
}

int main() {
  int a[MAX];
  int n, i, k;

  do {
    printf( "Nhap n [1, %d]: ", MAX - 1 );
    scanf( "%d", &n );
  } while ( n < 1 || n > MAX - 1 );

  printf( "Nhap %d phan tu:\n", n );
  for ( i = 0; i < n; ++i )
    scanf( "%d", a + i );
  printf( isSymmetrical( a, n ) ? "Doi xung" : "Khong doi xung" );

  printf( "\nNhap so lan can dich: " );
  scanf( "%d", &k );
  lshiftkstep( a, n, k );

  for ( i = 0; i < n; ++i )
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    printf( "%d ", a[i] );
  putchar( '\n' );
  return 0;
}
```

Sơ đồ thực hiện một bước dịch trái mảng của hàm `lshiftkstep()` được trình bày trong hình bên:

- Lưu tạm phần tử `a[0]` vào biến `t`.
- `n - 1` phần tử của mảng (kể từ `a[1]`) được sao chép dịch trái một phần tử, chồng lên `a[0]`, bằng hàm `memmove()`.
- `a[n - 1] = t`, nghĩa là bằng `a[0]` vừa bị "đẩy" khỏi đầu mảng.

Chú ý, nếu xoay vòng `n` lần, mảng sẽ trở lại như cũ, vì vậy với `k > n`, thực tế `k = k % n`.

Từ bài tập này trở đi, chúng ta thực hiện các yêu cầu của bài tập bằng cách gọi hàm để làm quen với việc xây dựng các hàm riêng lẻ.

Bài 72: (trang 21)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define MAX 100

void rshiftkstep( int a[], int n, int k ) {
  int i;
  for ( i = 0; i < k; ++i ) {
    int t = a[n - 1];
    memmove( a + 1, a, ( n - 1 ) * sizeof( *a ) );
    a[0] = t;
  }
}

int maxOdd( int a[], int n ) {
  int i, maxpos = -1;
  /* Tìm phần tử có trị lẻ đầu tiên */
  for ( i = 0; i < n; ++i )
    if ( a[i] % 2 ) break;
  if ( i < n ) {
    maxpos = i;
    for ( i = maxpos + 1; i < n; ++i )
      if ( a[i] % 2 && a[i] > a[maxpos] ) maxpos = i;
  }
  return maxpos;
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
int main() {
  int a[MAX];
  int n, i, k;
  srand( time( NULL ) );
  do {
    printf( "Nhap n [1, %d]: ", MAX - 1 );
    scanf( "%d", &n );
  } while ( n < 1 || n > MAX - 1 );
  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] = rand() % 201 - 100 );
  putchar( '\n' );

  if ( ( k = maxOdd( a, n ) ) >= 0 )
    printf( "Phan tu le lon nhat a[%d] = %d", k, a[k] );
  else printf( "Mang toan so chan" );

  printf( "\nNhap so lan can dich: " );
  scanf( "%d", &k );
  k %= n;
  rshiftkstep( a, n, k );
  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] );
  putchar( '\n' );
  return 0;
}
```

Sơ đồ thực hiện một bước dịch phải mảng của hàm `rshiftkstep()` được trình bày trong hình dưới, là hình ảnh đảo ngược của hàm `lshiftkstep()` trong bài 71 (trang 128):

- Lưu tạm phần tử `a[n-1]` vào biến `t`.
- `n-1` phần tử của mảng (kể từ `a[0]`) được sao chép dịch phải một phần tử, chồng lên mảng bắt đầu từ `a[1]`, bằng hàm `memmove()`.
- `a[0] = t`, nghĩa là bằng `a[n-1]` vừa bị "đẩy" khỏi cuối mảng.

Kỹ thuật tìm một phần tử thỏa yêu cầu từ một dãy phần tử đã được trình bày trong bài 66 (trang 121). Tuy nhiên, trong bài tập này cần chú ý:

- Tìm phần tử min, max của một mảng bao giờ cũng có kết quả, nên từ đầu ta có thể giả sử và gán `maxpos = 0` (chỉ số của phần tử đầu tiên). Tuy nhiên, khi tìm phần tử có trị lẻ lớn nhất, chưa chắc có kết quả; ví dụ với mảng chứa toàn số chẵn hoặc `0`. Vì vậy trước hết cần tìm phần tử có trị lẻ đầu tiên của mảng rồi mới tiến hành tìm phần tử lẻ có trị lớn nhất theo cách đã biết.

```
int maxOdd( int a[], int n ) {
  int i, maxpos = -1;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  for ( i = 0; i < n; ++i )
    if ( a[i] % n ) break;
  if ( i < n ) {
    maxpos = i;
    for ( i = maxpos + 1; i < n; ++i )
      if ( a[i] % 2 && a[i] > a[maxpos] ) maxpos = i;
  }
  return maxpos;
}
```

Nếu mảng không có trị lẻ, sau vòng lặp thứ nhất `i` sẽ bằng `n`, chương trình không vào thân của phát biểu `if` và `maxpos` trả về sẽ là trị khởi tạo `-1`.
Ngược lại, `maxpos` là vị trí của trị lẻ đầu tiên, ta tiếp tục tìm và so sánh các trị lẻ kế tiếp nếu có kể từ vị trí `maxpos + 1`.
Xây dựng hàm tìm kiếm trả về chỉ số của phần tử tìm được là phù hợp với tư duy theo chỉ số khi lập trình bằng C với mảng. Ưu điểm còn ở chỗ khi không tìm thấy phần tử thỏa yêu cầu, hàm trả về `-1`, ứng với "không vị trí nào".

Bài 73: (trang 22)

```
#include <stdio.h>
#define MAX 100

int main() {
  int a[MAX];
  int n, i, j;

  do {
    printf( "Nhap n [1, %d]: ", MAX - 1 );
    scanf( "%d", &n );
  } while ( n < 1 || n > MAX - 1 );

  printf( "Nhap %d phan tu:\n", n );
  for ( i = 0; i < n; ++i )
    scanf( "%d", a + i );

  for ( i = 0; i < n; ++i ) {
    for ( j = 0; j < i; ++j )
      if ( a[j] == a[i] ) break;
    if ( j == i ) printf( "%d ", a[i] );
  }
  putchar( '\n' );
  return 0;
}
```

Bài tập này dùng một kỹ thuật đơn giản để phát hiện và không xuất các phần tử có trị trùng lặp, ngoại trừ phần tử đầu tiên:
Xét phần tử `a[i]`, vòng lặp theo `j` bên trong sẽ duyệt tất cả các phần tử nằm *trước* vị trí `i`. Có hai trường hợp:
- Vòng lặp kiểm tra `j` ngắt nửa chừng do phát hiện một phần tử trùng với `a[i]`. Khi đó, `j < i` và đã có một phần tử có trị trùng `a[i]` được xuất trước đây, ta không xuất `a[i]` nữa.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

- Vòng lặp kiểm tra `j` chạy đến cuối (đến `i`) mà không phát hiện một phần tử nào trùng với `a[i]`. Khi đó, `j = i` và phần tử `a[i]` chưa xuất hiện lần nào, có thể xuất được.

Bài tập: Viết chương trình thực hiện những yêu cầu sau:

a. Tạo mảng một chiều `n` phần tử nguyên có giá trị nhập vào từ bàn phím. Giá trị của các phần tử thuộc đoạn `[0, n - 1]`.

b. In ra các phần tử trong mảng có trị phân biệt.

```
Nhap n [1, 99]: 10 ↵
Nhap 10 phan tu:
1 2 2 3 4 3 1 5 5 4 ↵
1 2 3 4 5
```

Lưu ý điều kiện "giá trị của các phần tử thuộc đoạn `[0, n - 1]`", đưa đến giải thuật:

- Phân phối trị `x` vào đúng phần tử có chỉ số `x`. Nghĩa là, nếu trị `x` tại `a[i]` không trùng với chỉ số phần tử mảng (`i`) chứa nó, nó sẽ được hoán chuyển với số tại `a[x]` cho đến khi điều kiện trên thỏa.
- Trị `i` có thể không có trong mảng, như vậy khi hoán chuyển đến một lúc nào đó, `x = a[i]` sẽ là một trị trùng lặp, `x = a[i]` sẽ trùng `x = a[x]`. Ta tránh trường hợp này bằng cách ngắt vòng lặp khi `a[i] = a[a[i]]`.
- Sau khi thực hiện phân phối, in các trị phân biệt khi thỏa điều kiện `a[i]` bằng `i`.

```
for ( i = 0; i < n; ++i )
  while ( a[i] != i ) {
    if ( a[i] == a[a[i]] ) break;
    t = a[i]; a[i] = a[t]; a[t] = t;
  }
for ( i = 0; i < n; ++i )
  if ( a[i] == i ) printf( "%d ", a[i] );
```

Ta nhận thấy, trường hợp `break` cũng là trường hợp phát hiện các trị trùng lặp. Nếu yêu cầu là in ra các trị bị trùng lặp, in trị `a[i]` trước khi `break`.

Bài 74: (trang 22)

```
#include <stdio.h>
#define MAX 100

int main() {
  int a[MAX];
  int n, i, j, count;

  do {
    printf( "Nhap n [1, %d]: ", MAX - 1 );
    scanf( "%d", &n );
  } while ( n < 1 || n > MAX - 1 );
  printf( "Nhap %d phan tu:\n", n );
  for ( i = 0; i < n; ++i )
    scanf( "%d", a + i );

  for ( i = 0; i < n; ++i ) {
    for ( count = j = 0; j < n; ++j )
      if ( a[j] == a[i] )
        if ( j < i ) break;
        else count++;
    if ( count ) printf( "%d[%d] ", a[i], count );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  }
  putchar( '\n' );
  return 0;
}
```

Cách giải thường gặp là dùng một mảng phụ để lưu tần suất của từng phần tử.Vấn đề chủ yếu của bài tập này là chỉ đếm số lần xuất hiện của một phần tử nếu nó xuất hiện lần đầu tiên trong mảng. Nếu nhận ra một phần tử *đã xuất hiện* trong mảng, ta phải bỏ qua, không đếm nữa. Cách phát hiện một phần tử đã xuất hiện là mở rộng của bài tập 73 (trang 131):

Xét phần tử `a[i]`, vòng lặp theo `j` bên trong sẽ duyệt đếm tất cả các phần tử trùng với `a[i]` có trong mảng `a`. Vòng lặp này có đặc điểm:

- Khởi tạo biến đếm `count` bằng `0` trước khi duyệt.
- Khi gặp một phần tử trùng với `a[i]`, vòng lặp kiểm tra: nếu `j < i` thì nghĩa là phát hiện được một phần tử có trị trùng `a[i]` *đã xuất hiện* trước đây, vòng lặp ngắt ngay và như vậy `count` vẫn bằng `0`. Ngược lại vòng lặp vẫn đếm bình thường.

Với cách kiểm tra như trên, chỉ có những phần tử xuất hiện lần đầu tiên được đếm, `count` của chúng khác `0`, và được xuất ra.

Bài 75: (trang 22)

```
#include <stdio.h>
#define MAX 100

int main() {
  int a[MAX];
  int n, i, j, count, max, min, pos;

  do {
    printf( "Nhap n [1, %d]: ", MAX - 1 );
    scanf( "%d", &n );
  } while ( n < 1 || n > MAX - 1 );
  printf( "Nhap %d phan tu:\n", n );
  for ( i = 0; i < n; ++i )
    scanf( "%d", a + i );

  printf( "Phan tu xuat hien nhieu nhat: " );
  pos = max = 0;
  for ( i = 0; i < n; ++i ) {
    for ( count = 1, j = i + 1; j < n; ++j )
      if ( a[j] == a[i] ) count++;
    if ( count > max ) { max = count; pos = i; }
  }
  printf( "%d[%d]\n", a[pos], max );

  printf( "Phan tu xuat hien it nhat: " );
  pos = 0; min = n;
  for ( i = 0; i < n; ++i ) {
    for ( count = j = 0; j < n; ++j )
      if ( a[j] == a[i] )
        if ( j < i ) break;
        else count++;
    if ( count && count < min ) { min = count; pos = i; }
  }
  printf( "%d[%d]\n", a[pos], min );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  return 0;
}
```

Trong bài tập 74 (trang 132), khi tính tần suất các phần tử, ta phải giải quyết vấn đề tránh tính tần suất của các ký tự *đã xuất hiện*. Với yêu cầu tìm phần tử có tần suất lớn nhất, ta không cần chú ý vấn đề này, vì tần suất của phần tử đã xuất hiện vẫn nhỏ hơn tần suất của phần tử này khi được tính lần đầu.
Ta khởi tạo trị `max` bằng `0`, tính tần suất cho mọi vị trí dù phần tử tại vị trí đó đã xuất hiện, rồi so sánh với `max` để tìm tần suất lớn nhất.
Với yêu cầu tìm phần tử có tần suất nhỏ nhất, ta khởi tạo trị `min` bằng kích thước toàn mảng. Sau đó tìm tần suất `count` của từng phần tử như trong bài 74 (trang 132), rồi so sánh với `min` để tìm tần suất nhỏ nhất.
Chú ý khi gặp phần tử trùng lặp ta không tính tần suất, ngắt vòng lặp nên `count = 0`, vì vậy đừng quên kiểm tra `count` khác `0` trước khi so sánh `count` với `min`.

Bài 76: (trang 22)

```
#include <stdio.h>
#define MAX 100

int main() {
  int a[MAX], max, oldmax, pos;
  int n, i, j, count;

  do {
    printf( "Nhap n [1, %d]: ", MAX - 1 );
    scanf( "%d", &n );
  } while ( n < 1 || n > MAX - 1 );

  printf( "Nhap %d phan tu:\n", n );
  for ( i = 0; i < n; ++i )
    scanf( "%d", a + i );
  printf( "Cac phan tu xuat hien nhieu nhat:\n" );
  oldmax = 0;
  pos = -1;
  do {
    max = 0;
    for ( i = pos + 1; i < n; ++i ) {
      for ( count = 1,j = i + 1; j < n; ++j )
        if ( a[j] == a[i] ) count++;
      if ( count > max ) { max = count; pos = i; }
    }
    if ( max >= oldmax ) {
      printf( "%d[%d] ", a[pos], max );
      oldmax = max;
    }
  } while ( max == oldmax );
  putchar( '\n' );
  return 0;
}
```

Trong bài tập 75 (trang 133) ta chỉ tìm được phần tử có tần suất lớn nhất *đầu tiên*. Để tìm tất cả các phần tử có cùng tần suất lớn nhất ta phải:
- Đưa việc tính toán tần suất lớn nhất trên vào một *vòng lặp tìm tiếp* (vòng lặp `do while` ở trên), vòng lặp này được thiết kế hoạt động với sự giúp đỡ của 2 biến:

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Biến `oldmax` lưu tần suất lớn nhất tìm được lần trước, với lần tìm đầu tiên `oldmax` khởi tạo bằng `0`.
Biến `pos` lưu vị trí của phần tử có tần suất lớn nhất tìm được lần trước, với lần tìm đầu tiên `pos` khởi tạo bằng `-1`. Khi tiếp tục tìm tần suất lớn nhất mới, ta sẽ bắt đầu sau vị trí `pos` này.
- Với mỗi tần suất `max` lớn nhất mới tìm được ta so sánh với `oldmax`. Dù `max` không bao giờ lớn hơn `oldmax` nhưng vẫn phải dùng điều kiện so sánh `>=` cho trường hợp đầu tiên.
Nếu đã tìm được `max` mới có cùng tần suất lớn nhất với `oldmax` thì xuất ra; nếu không vòng lặp tìm tiếp sẽ chấm dứt.

Bài 77: (trang 22)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define MAX 100

double invSum( int a[], int n ) {
  int i;
  double sum = 0.0;
  for ( i = 0; i < n; ++i )
    if ( a[i] ) sum += 1.0/a[i];
  return sum;
}

void rmLeftOdd( int a[], int* n ) {
  int i;
  for ( i = 0; i < *n - 2; ++i )
    if ( ( a[i] % 2 ) ) {
      memmove( a+i+1, a+i+2, ( *n-i-2 ) * sizeof( *a ) );
      ( *n )--;
    }
  if ( ( a[*n - 2] % 2 ) ) ( *n )--;
}

int main() {
  int a[MAX];
  int n, i;

  srand( time( NULL ) );
  do {
    printf( "Nhap n [1, %d]: ", MAX - 1 );
    scanf( "%d", &n );
  } while ( n < 1 || n > MAX - 1 );

  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] = rand() % 201 - 100 );
  printf( "\nTong nghich dao: %g\n", invSum( a, n ) );
  rmLeftOdd( a, &n );
  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] );
  putchar( '\n' );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  return 0;
}
```

Khi thao tác trên một mảng bạn cần chú ý đến vấn đề kiểm soát các phần tử biên. Thuật toán áp dụng với các phần tử của mảng có thể không áp dụng đúng với các phần tử của biên mảng. Bạn phải kiểm tra để giải quyết các ngoại lệ này.

Trong hàm `rmLeftOdd()`, khi phát hiện phần tử `a[i]` lẻ, ta chuyển đoạn mảng từ `a[i+2]` chồng lên đoạn mảng từ `a[i+1]`. Nhưng nếu `a[n-2]` lẻ, chỉ còn một phần tử sau `a[n-2]`, không thể áp dụng việc chồng mảng mà chỉ đơn giản giảm `n`, không chú ý đến phần tử `a[n-1]`, tương đương với loại bỏ nó.

Truyền tham chiếu bằng con trỏ (gọi tắt là truyền bằng con trỏ) là cách của C dùng thay đổi tham số được truyền đến hàm. Xem ví dụ dưới:

Truyền bằng tham trị

```
#include <stdio.h>

void func( int x ) {
  x = x * 2;
}

int main() {
  int a = 3;
  func( a );
  /* a không đổi a = 3 */
  printf( "%d\n", a );
  return 0;
}
```

COPY

a được sao chép đến **x**: **int x = a** sự thay đổi của **x** không liên quan đến **a**

Truyền bằng con trỏ

```
#include <stdio.h>

void func( int *x ) {
  *x = *x * 2;
}

int main() {
  int a = 3;
  func( &a );
  /* a đã thay đổi a = 6 */
  printf( "%d\n", a );
  return 0;
}
```

COPY

"khai báo" con trỏ **x** chỉ đến **a**: **int* x = &a** thay đổi ***x** nghĩa là "gián tiếp" thay đổi **a**

- Hàm `func()` bên trái không thể thay đổi biến `a` của hàm `main()` do khác tầm vực (scope). Truyền bằng tham trị thực chất là sao chép biến `a` của hàm `main()` vào biến cục bộ `x` của hàm `func()`.
- Hàm `func()` bên phải được truyền tham số bằng con trỏ, thực chất là "khai báo" một con trỏ `x` chỉ đến biến `a` của hàm `main()`. Thông qua con trỏ này, hàm `func()` có thể thay đổi "gián tiếp" biến `a` của hàm `main()`: khi đó, thay đổi `*x` nghĩa là đang thay đổi "gián tiếp" biến `a`.

C++ giải quyết vấn đề này bằng tham chiếu, khái niệm này không có trong C.

Truyền tham chiếu bằng con trỏ còn thường dùng khi muốn trả về nhiều hơn một trị, ví dụ hàm `swap()` dùng hoán chuyển trị của hai biến, hoặc khi muốn tự động thay đổi một trị nào đó do hàm mà không cần chú ý cập nhật lại.

Bạn ôn lại việc sử dụng các toán tử dùng thao tác với con trỏ:

- Toán tử `&`: toán tử lấy địa chỉ của một biến để "đặt" vào con trỏ. Kết quả là con trỏ sẽ chỉ đến biến đó.
- Toán tử `*`: (dereference) toán tử lấy nội dung của một biến do con trỏ chỉ đến. Dùng toán tử này để đọc/ghi "gián tiếp" biến do con trỏ chỉ đến.

Bài 78: (trang 23)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define MAX 100
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
void sort( int a[], int n ) {
  int i, j;
  for ( i = 0; i < n - 1; ++i )
    for ( j = i + 1; j < n; ++j )
      if ( a[i] > a[j] )
        { int t = a[i]; a[i] = a[j]; a[j] = t; }
}

void insertOrder( int a[], int* n, int x ) {
  int i = 0;
  while ( i < *n && a[i] < x ) i++;
  if ( i < *n )
    memmove( a + i + 1, a + i, ( *n - i ) * sizeof( *a ) );
  a[i] = x;
  ( *n )++;
}

int main() {
  int a[MAX], x;
  int n, i;

  srand( time( NULL ) );
  do {
    printf( "Nhap n [1, %d]: ", MAX - 1 );
    scanf( "%d", &n );
  } while ( n < 1 || n > MAX - 1 );
  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] = rand() % 201 - 100 );

  printf( "\nMang sap xep tang:\n" );
  sort( a, n );
  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] );
  printf( "\nNhap x: " );

  scanf( "%d", &x );
  insertOrder( a, &n, x );

  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] );
  putchar( '\n' );
  return 0;
}
```

Hàm `insertOrder()` thực hiện hai thao tác; do vấn đề kiểm soát phần tử biên, cả hai thao tác đều phải chú ý đến các ngoại lệ:

- Tìm đến vị trí cần chèn: nếu các phần tử của mảng đều nhỏ hơn trị cần chèn, vòng lặp tìm kiếm sẽ vượt quá cuối mảng, cần phải kiểm soát bằng điều kiện phụ `i < n`.
- Dịch chuyển một phần mảng để chèn: nếu các phần tử của mảng đều nhỏ hơn trị cần chèn, vị trí cần chèn là vị trí mới cuối mảng, vì vậy không cần phải dịch chuyển mảng, ta cũng kiểm soát trường hợp này bằng điều kiện phụ `i < n`.

Có thể giải quyết vấn đề trên bằng cách tạo "phần tử ma" (ghost node) thêm vào cuối mảng, phần tử này chứa trị lớn hơn bất kỳ phần tử nào khác của mảng. Khi đó hàm `insertOder()` được viết gọn hơn:

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
void insertOrder( int a[], int* n, int x ) {
  int i = 0;
  a[( *n )++] = 101;
  while ( a[i] < x ) i++;
  memmove( a + i + 1, a + i, ( *n - i ) * sizeof( *a ) );
  a[i] = x;
}
```

Với mảng đã sắp xếp có số phần tử lớn, người ta thường dùng giải thuật tìm kiếm nhị phân để xác định rất nhanh vị trí chèn:

```
void insertOrder( int a[], int* n, int x ) {
  int left, right, mid;
  a[( n )++] = 101; /* ghost node */
  left = 0;     /* biên trái */
  last = *n;        /* biên phải */
  while ( left <= last ) {
    mid = ( left + last ) / 2;
    /* vị trí của x trong nửa đầu đoạn cần tìm ⇒ điều chỉnh biên trái
       ngược lại, điều chỉnh biên phải */
    if ( x < a[mid] ) last = mid - 1;
    else              left = mid + 1;
  }
  /* left bây giờ là vị trí đúng của x */
  memmove( a+left+1, a+left, ( *n - left )*sizeof( *a ) );
  a[first] = x;
}
```

Bài 79: (trang 23)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <math.h>

int* myAlloc( int n ) {
  return ( int* )calloc( n, sizeof( int ) );
}

void printArr( int *a, int n ) {
  int i;
  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] );
  putchar( '\n' );
}

int nearX( int *a, int n, int x ) {
  int i, pos;
  int min = 201;
  for ( i = 0; i < n; ++i )
    if ( abs( a[i] - x ) < min )
      { min = abs( a[i] - x ); pos = i; }
  return pos;
}

int insertRNeg( int **a, int *n ) {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  int i, c, *p;
  for ( c = i = 0; i < *n; ++i )
    c += ( (*a)[i] < 0 );
  if ( c ) {
    p = ( int* )realloc( *a, ( *n + c ) * sizeof( int ) );
    if ( !p ) return 1;
    *a = p;
    int k = *n + c - 1;
    for ( i = *n - 1; i >= 0; --i ) {
      if ( (*a)[i] < 0 ) (*a)[k--] = 1;
      (*a)[k--] = (*a)[i];
    }
    *n += c;
  }
  return 0;
}

int main() {
  int n, i;
  int *a, x;

  srand( time( NULL ) );
  do { printf( "Nhap n (n > 0): " );
    scanf( "%d", &n );
  } while ( n < 1 );

  a = myAlloc( n );
  if ( !a )
    { printf( "Loi cap phat\n" ); return 1; }

  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] = rand() % 201 - 100 );

  printf( "\nNhap x: " );
  scanf( "%d", &x );
  printf( "So gan x nhat: %d\n", a[nearX( a, n, x )] );

  if ( insertRNeg( &a, &n ) )
    { printf( "Loi cap phat\n" ); return 2; }
  printArr( a, n );
  free( a );
  return 0;
}
```

Bài tập này minh họa các thao tác cần thực hiện khi làm việc với mảng cấp phát động, ôn lại các hàm dùng cấp phát động vùng nhớ trên heap (`stdlib.h`):

- `void *malloc( size_t size )`: trả về con trỏ chỉ đến byte đầu tiên của vùng nhớ có kích thước *size* được cấp phát trên heap. Nếu không đủ vùng nhớ trên heap cho yêu cầu cấp phát, trả về con trỏ `NULL`.
- `void *calloc( size_t num, size_t size )`: trả về con trỏ chỉ đến byte đầu tiên của vùng nhớ cấp phát cho một mảng có *num* phần tử, mỗi phần tử có kích thước *size*. Tất cả các bit của vùng nhớ cấp phát được đặt bằng `0`. Nếu không đủ vùng nhớ trên heap cho yêu cầu cấp phát, trả về con trỏ `NULL`.
- `void *realloc( void *ptr, size_t size )`: thay đổi kích thước của vùng nhớ đã được cấp phát trước đây (chỉ bởi con trỏ *ptr*) thành kích thước *size* (lớn hơn hoặc

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

nhỏ hơn kích thước cũ). Một con trỏ chỉ đến vùng nhớ cấp phát được trả về do có sự sao chép nội dung vùng nhớ cũ sang vùng nhớ mới.

- `void free( void *ptr )`: giải phóng vùng nhớ cấp phát trên heap chỉ bởi con trỏ *ptr* cho những nhu cầu cấp phát khác.

Hàm `insertRNeg()` có hai tham số đều thay đổi sau khi gọi hàm nên được truyền bằng con trỏ (xem bài 77, trang 135): con trỏ `a` quản lý mảng sẽ được cấp phát lại và số phần tử `n` của mảng sẽ thay đổi khi chèn thêm `1` vào mảng.

Thao tác chèn vào mảng có thể thực hiện như bài 78 (trang 136), bằng hàm `memmove()` của `string.h`:

```
for ( i = *n; i > 0; --i )
  if ( (*a)[i - 1] < 0 ) {
    memmove( *a+i+1, *a+i, (*n - i) * sizeof( int ) );
    (*a)[i] = 1;
    (*n)++;
  }
```

tuy nhiên, do biết trước số phần tử mới của mảng (`n + c`, `c` là số các số nguyên âm có trong mảng), nên việc chèn vào mảng được thực hiện dễ hiểu như sau:

```
/* k chỉ vị trí cuối mảng mới */
k = n + c - 1;
for ( i = *n - 1; i >= 0; --i ) {
/* sao chép từng phần tử từ mảng cũ (a) sang mảng mới (vẫn là a) theo thứ
tự từ cuối mảng ra trước, nếu gặp phần tử âm thì sao chép 1 trước khi sao
chép phần tử đó */
  if ( (*a)[i] < 0 ) (*a)[k--] = 1;
  (*a)[k--] = (*a)[i];
}
*n += c;          /* cập nhật số phần tử mới của mảng */
```

Cách viết `(*a)[i]` rõ ràng dễ nhầm lẫn hơn cách viết theo địa chỉ: `(*a + i)`.

Bài 80: (trang 23)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 100

int subArrPos( int a[], int b[], int n, int m ) {
  int i, j;
  i = j = 0;
  do {
    if ( a[i] == b[j] ) { i++; j++; }
    else { i = i - j + 1; j = 0; }
  } while ( i <= n - m && j < m );
  return ( j >= m ) ? i - m : -1;
}

int checkNegInt( int a[], int n ) {
  int i;
  for ( i = n - 1; i >= 0; --i )
    if ( a[i] < 0 ) break;
  return i;
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
int main() {
  int  a[MAX], b[MAX], i;
  int n, m;
  do {
    printf( "Nhap n [1, %d]: ", MAX - 1 );
    scanf( "%d", &n );
  } while ( n < 1 || n > MAX - 1 );
  printf( "Nhap %d phan tu mang A:\n", n );
  for ( i = 0; i < n; ++i )
    scanf( "%d", a + i );

  do {
    printf( "Nhap m [1, %d]: ", n );
    scanf( "%d", &m );
  } while ( m < 1 || m > n );
  printf( "Nhap %d phan tu mang B:\n", m );
  for ( i = 0; i < m; ++i )
    scanf( "%d", b + i );

  i = subArrPos( a, b, n, m );
  if ( i != -1 ) printf( "B co trong A tai: A[%d]\n", i );
  else printf( "B khong thay trong A\n" );

  i = checkNegInt( a, n );
  if ( i != -1 ) printf( "So nguyen am cuoi: %d\n", a[i] );
  else printf( "Mang khong co so nguyen am\n" );
  return 0;
}
```

Hàm `subArrPos()` thực chất là tìm một "run" `B` trong mảng `A`, xem bài 69 (trang 126). Hàm `subArrPos()` hoạt động như sau:

```
i = j = 0; /* i dùng duyệt a, j dùng duyệt b */
do {
  /* nếu a[i] bằng b[j] thì tăng i và j rồi kiểm tra tiếp */
  if ( a[i] == b[j] ) { i++; j++; }
  /* nếu không, j chỉ về đầu b, i chỉ đến vị trí bắt đầu kiểm tra lần trước + 1 */
  else { i = i - j + 1; j = 0; }
/*
  có 2 trường hợp kết thúc:
   + đoạn còn lại trên a ngắn hơn m: n - i < m ⇒ điều kiện vòng lặp i <= n - m
   + đã kiểm tra 'run' b thành công: j >= m ⇒ điều kiện vòng lặp j < m
*/
} while ( i <= n - m && j < m );
/* nếu thành công, trả về vị trí bắt đầu 'run' b trên a,
   ngược lại, trả về -1 (không vị trí nào) */
return ( j >= m ) ? i - m : -1;
```

Chú ý cách trả về của một hàm tìm vị trí trong một mảng. Nếu không tìm thấy vị trí theo yêu cầu, trả về `-1`, chỉ số không thể có trong mảng.

Bài 81: (trang 24)

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

int* myAlloc( int n ) {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  return ( int* ) calloc( n, sizeof( int ) );
}

int* initArr( int n ) {
  int i;
  int* a = myAlloc( n );
  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] = rand() % 201 - 100 );
  putchar( '\n' );
  return a;
}

void printArr( int *a, int n ) {
  int i;
  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] );
  putchar( '\n' );
}

int* sort( int *a, int n ) {
  int i, j;
  for ( i = 0; i < n - 1; ++i )
    for ( j = i + 1; j < n; ++j )
      if ( a[i] > a[j] )
        { int t = a[i]; a[i] = a[j]; a[j] = t; }
  return a;
}

int* mergeArrDesc( int *a, int *b, int n, int m ) {
  int i = n - 1;
  int j = m - 1;
  int* c = myAlloc( n + m );
  if ( c ) {
    int k = 0;
    while ( i >= 0 && j >= 0 )
      c[k++] = ( a[i] > b[j] ) ? a[i--] : b[j--];
    while ( i >= 0 ) c[k++] = a[i--];
    while ( j >= 0 ) c[k++] = b[j--];
  }
  return c;
}

int main() {
  int *a, *b;
  int n, m;

  srand( time( NULL ) );
  do {
    printf( "Nhap so phan tu mang A va B (n, m > 0): " );
    scanf( "%d%d", &n, &m );
  } while ( n < 1 || m < 1 );

  a = myAlloc( n );
  b = myAlloc( m );
  if ( !a || !b ) printf( "Loi cap phat\n" );
  else {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    a = initArr( n );
    b = initArr( m );
    printf( "Mang A sap tang: " ); printArr( sort( a, n ), n );
    printf( "Mang B sap tang: " ); printArr( sort( b, m ), m );
    printf( "Tron A va B thanh C sap giam:\n" );
    int *c = mergeArrDesc( a, b, n, m );
    if ( !c ) printf( "Loi cap phat\n" );
    else { printArr( c, n + m ); free( c ); }
  }
  if ( a ) free( a );
  if ( b ) free( b );
  return 0;
}
```

Thao tác được yêu cầu, thường gọi là "trộn" (merge), thực hiện bằng hàm `mergeArrDesc()`. Hàm này tiến hành duyệt các mảng `a` và `b` đã sắp xếp *từ cuối đến đầu mảng*, lấy các phần tử thích hợp chuyển vào mảng `c`. Có hai giai đoạn:

- So sánh `a[i]` và `b[j]`, phần tử nào lớn hơn sẽ được đưa vào mảng `c`. Thực hiện cho đến khi chuyển hết phần tử của một trong hai mảng.
- Chuyển tất cả các phần tử còn lại của mảng kia vào mảng `c`.

Nếu yêu cầu trộn mảng `a` và `b` thành mảng `c` *sắp xếp tăng*, có thể tiến hành tương tự hoặc dùng phương pháp "phần tử ma" (ghost node) để bỏ đi giai đoạn sau, xem thêm bài 78 (trang 23):

```
int* mergeArrAsc( int *a, int *b, int n, int m ) {
  int i, j, k;
  a[n] = b[m] = 101;
  i = j = k = 0;
  int* c = myAlloc( n + m );
  if ( c ) {
    while ( k <= n + m - 1 )
      c[k++] = ( a[i] < b[j] ) ? a[i++] : b[j++];
  }
  return c;
}
```

Chú ý phải cấp phát thêm `1` phần tử cho mảng `a` và mảng `b`, cho "phần tử ma" này.

Bài 82: (trang 24)

```
#include <stdio.h>
#define MAX 100

void output( int a[], int n, char* s ) {
  int i;
  printf( "%s: { ",s );
  for ( i = 0; i < n; ++i )
    printf( "%d%s", a[i], ( i == n - 1 ) ? " }\n" : ", " );
}

int isMember( int a[], int n, int x ) {
  int i;
  for ( i = 0; i < n; ++i )
    if ( x == a[i] ) return 1;
  return 0;
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
int main() {
  int a[MAX], x;
  int n, i;

  printf( "Nhap khong qua 100 phan tu (Ctrl+Z de dung)\n" );
  for ( n = i = 0; scanf( "%d", &x ) == 1 && i < MAX; ++i )
    if ( !isMember( a, n, x ) ) a[n++] = x;
  output( a, n, "Tap hop A" );
  return 0;
}
```

Hàm `isMember()` là trung tâm của bài tập này, nhằm xác định xem một số `x` có thuộc về tập hợp (mảng) `a` không. Trên cơ sở của hàm này, bất kỳ số mới nhập nào cũng được kiểm tra xem đã tồn tại chưa. Nếu chưa, số mới nhập mới được đưa vào mảng và số phần tử `n` của mảng mới được tăng lên.
Chú ý cách kiểm tra nhập với hàm `scanf()`, xem bài 30 (trang 91).

Bài 83: (trang 24)

```
#include <stdio.h>
#define MAX 100

void output( int a[], int n, char* s ) {
  int i;
  printf( "%s: { ",s );
  for ( i = 0; i < n; ++i )
    printf( "%d%s", a[i], ( i == n-1 ) ? " }\n" : ", " );
}

int isMember( int a[], int n, int x ) {
  int i;
  for ( i = 0; i < n; ++i )
    if ( x == a[i] ) return 1;
  return 0;
}

int AandB( int a[], int n, int b[], int m, int c[] ) {
  int i, j, k;
  for ( k = i = 0; i < n; ++i )
    for ( j = 0; j < m; ++j )
      if ( b[j] == a[i] )
        c[k++] = a[i];
  return k;
}

int AorB( int a[], int n, int b[], int m, int c[] ) {
  int i, j, k;
  i = j = k = 0;
  while ( i < n ) c[k++] = a[i++];
  while ( j < m )
    if ( !isMember( a, n, b[j] ) )
      c[k++] = b[j++];
    else j++;
  return k;
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
int AdecB( int a[], int n, int b[], int m, int c[] ) {
  int i, k;
  for ( k = i = 0; i < n; ++i )
    if ( !isMember( b, m, a[i] ) )
      c[k++] = a[i];
  return k;
}

int main() {
  int a[] = { 1, 2, 3, 5 };
  int b[] = { 1, 3, 6, 7 };
  int c[MAX];
  int n, m, k;

  n = sizeof a / sizeof *a;
  m = sizeof b / sizeof *b;

  output( a, n, "Tap hop A" );
  output( b, m, "Tap hop B" );

  k = AandB( a, n, b, m, c );
  output( c, k, "C = A * B" );

  k = AorB( a, n, b, m, c );
  output( c, k, "C = A + B" );

  k = AdecB( a, n, b, m, c );
  output( c, k, "C = A \\ B");
  return 0;
}
```

Với hàm `isMember()` (xem bài 82, trang 143) các phép toán trên tập hợp được thực hiện dễ dàng:

- `a ∩ b`: hai vòng lặp lồng nhau duyệt tất cả các cặp (`a[i]`, `b[j]`), cặp nào thỏa điều kiện `a[i] = b[j]` sẽ được đưa vào `c`.
- `a ∪ b`: đầu tiên toàn bộ các phần tử của `a` được chuyển vào `c`. Tiếp theo, phần tử nào của `b` và không là thành viên của `a` (trừ đi phần giao `a ∩ b`) được chuyển vào `c`.
- `a \ b`: phần tử nào của `a` và không là thành viên của `b` (trừ đi phần giao `a ∩ b`) sẽ được chuyển vào `c`.

Bài 84: (trang 24)

```
#include <stdio.h>
#include <string.h>
#define MAX 100

void insertOrder( int* a, int *n, int x ) {
  int i = 0;
  while ( i < *n && a[i] > x ) i++;
  if ( i < *n )
    memmove( a + i + 1, a + i, ( *n - i ) * sizeof( *a ) );
  a[i] = x;
  ( *n )++;
}

int tableReadAndSort( int* a ) {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  int n, x;
  for ( n = 0; scanf( "%d", &x ) == 1 && n < MAX; )
    insertOrder( a, &n, x );
  return n;
}

int main() {
  int a[MAX];
  int n, i;

  printf( "Nhap khong qua 100 phan tu (Ctrl+Z de dung)\n" );
  n = tableReadAndSort( a );
  for ( i = 0; i < n; ++i )
    printf( "%d ", a[i] );
  putchar( '\n' );
  return 0;
}
```

Hàm `insertOrder()` đã được giải thích trong bài 78 (trang 136), nhằm chèn một phần tử vào một mảng đã sắp xếp mà vẫn giữ nguyên tính chất sắp xếp của mảng. Sau khi gọi hàm này, số phần tử của mảng được truyền như tham số đến hàm sẽ thay đổi. Vì vậy trong hàm `tableReadAndSort()`, vòng lặp *không cần cập nhật* biến đếm `n` (biểu thức thứ ba của vòng lặp `for`), do đã được cập nhật bởi hàm `insertOrder()`.

Bài 85: (trang 25)

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

int main() {
  int a[4][4];
  int* b = &a[0][0];
  int i, j, t, n = 4;

  printf( "Mang goc:\n" );
  srand( time( NULL ) );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%5d", a[i][j] = rand() % 201 - 100 );

  for ( i = 0; i < n * n - 1; ++i )
    for ( j = i + 1; j < n * n; ++j )
      if ( b[i] > b[j] )
        { t = b[i]; b[i] = b[j]; b[j] = t; }

  printf( "Mang sau khi sap xep:\n" );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%5d", a[i][j] );
  return 0;
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Nếu xem mảng hai chiều như mảng một chiều tương tự hình dưới; và nếu các phần tử trong mảng một chiều được sắp xếp tăng dần thì "hình ảnh" hai chiều của chúng sẽ thỏa điều kiện bài tập trên: mỗi dòng tăng từ trái sang phải và mỗi cột tăng lên từ trên xuống bên.

Vì C xem mảng hai chiều như là "mảng của mảng", nói cách khác trong C mảng nào cũng là mảng một chiều. Vì vậy, mảng hai chiều được cấp phát trong vùng nhớ thực chất là một khối liên tục giống mảng một chiều như hình dưới.

Như vậy:

- Nếu quản lý *bằng chỉ số* (ví dụ `a[i][j]`), ta đã làm việc với mảng `a` như là mảng hai chiều kích thước `n x n`.
- Nếu quản lý *bằng con trỏ* `b` chỉ đến phần tử đầu tiên của mảng `a` (`int *b = &a[0][0]`), ta đã làm việc với mảng `a` như là mảng một chiều `b` kích thước `n x n`. Thao tác trên mảng một chiều `b` này rất bình thường do khả năng *gán chỉ số cho con trỏ* quản lý mảng của C. Trên cơ sở đó, bài tập trên được thực hiện như sau:
- Khởi tạo mảng: xem như khởi tạo mảng hai chiều `a`.
- Sắp xếp tăng: xem như sắp xếp mảng một chiều `b`.
- Xuất mảng: xem như xuất mảng hai chiều `a`.

Chú ý phải khởi tạo trước: `int* b = &a[0][0];`
hoặc để bảo đảm kiểu dữ liệu: `int* b = (int*)a;`

Bài tập trên đã cho *chính xác kích thước* của mảng hai chiều. Trong trường hợp chưa biết kích thước mảng, ta phải khai báo mảng tĩnh lớn hơn nên "hình ảnh" một chiều của chúng trong vùng nhớ sẽ sai lệch (xem hình dưới).

Khi đó, cách lập trình sẽ khác. Ta có nhu cầu sao chép mảng hai chiều thành mảng một chiều để tiện xử lý. Thao tác sao chép như sau, xem hình trên để hiểu rõ trình tự sao chép:

```
#define MAX 20
/* ... */
int a[MAX][MAX], b[MAX * MAX];
int n, i, j;
/* ... */
/* sao chép sang mảng 1 chiều */
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
for ( i = 0; i < n; ++i )
  memmove( &b[i * n], &a[i][0], n * sizeof( **a ) );

/* sao chép trở lại mảng 2 chiều */
for ( i = 0; i < n; ++i )
  memmove( &a[i][0], &b[i * n], n * sizeof( **a ) );
```

Tuy nhiên, vẫn có cách thực hiện bài tập mà không cần khai báo thêm mảng một chiều phụ.

Xét tổng quát với mảng hai chiều kích thước `n x m`:

- Hình dưới mô tả ánh xạ giữa một phần tử của mảng hai chiều với phần tử tương ứng trong mảng một chiều, nếu chuyển mảng hai chiều thành một chiều:

- Giả sử mảng hai chiều `a` đã được chuyển thành mảng một chiều `b`, ta sắp xếp mảng `b` tăng dần bằng sắp xếp kiểu chọn:

```
int k, l;
for ( k = 0; k < n * m - 1; ++k )
  for ( l = k + 1; l < n * m; ++l )
    if ( b[k] > b[l] ) {
      int t = b[k];
      b[k] = b[l];
      b[l] = t;
    }
```

- Thay thế `b[k]` và `b[l]` bằng phần tử tương ứng trong mảng hai chiều, xem công thức ánh xạ trong hình trên:

```
int k, l;
for ( k = 0; k < n * m - 1; ++k )
  for ( l = k + 1; l < n * m; ++l )
    if ( a[k / m][k % m] > a[l / m][l % m] ) {
      int t = a[k / m][k % m];
      a[k / m][k % m] = a[l / m][l % m];
      a[l / m][l % m] = t;
    }
```

Khi ma trận được sắp xếp như trên, việc tìm kiếm một trị trong ma trận có thể được tiến hành theo cách tìm kiếm nhị phân.

Bài 86: (trang 25)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 20
#define swap( a, b ) { int t = a; a = b; b = t; }

int main() {
  int a[MAX][MAX], b[MAX];
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  int n, i, j, k;

  printf( "Nhap n (n < 20): " );
  scanf( "%d", &n );

  srand( time( NULL ) );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%5d", a[i][j] = rand() % 201 - 100 );

  /* lưu tổng trị các cột vào mảng b */
  for ( i = 0; i < n; ++i ) {
    int t;
    for ( t = j = 0; j < n; ++j )
      t += a[i][j];
    b[i] = t;
  }

  /* sắp xếp mảng b, kéo theo sắp xếp các cột mảng a */
  for ( i = 0; i < n - 1; ++i )
    for ( j = i + 1; j < n; ++j )
      if ( b[i] > b[j] ) {
        /* hoán chuyển b[i] với b[j] */
        swap( b[i], b[j] );
        /* hoán chuyển cột a[][i] với cột a[][j] */
        for ( k = 0; k < n; ++k )
          swap( a[k][i], a[k][j] );
      }

  printf( "Mang sau khi sap xep:\n" );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%5d", a[i][j] );
  return 0;
}
```

Trước hết ta lưu tổng trị các cột của mảng hai chiều `a` (kích thước `n x n`) vào mảng một chiều `b` (kích thước `n`).
Sau đó ta sắp xếp tăng các phần tử trong `b`: khi hoán chuyển hai phần tử `b[i]` (chứa tổng trị cột `a[][i]`) và `b[j]` (chứa tổng trị cột `a[][j]`) trong `b`, ta cũng đồng thời hoán chuyển hai cột `a[][i]` và `a[][j]` tương ứng trong `a`.

Bài 87: (trang 25)

```
#include <stdio.h>
#define MAX 20

int triMatrix( int a[][MAX], int n ) {
  int i, j;
  for ( i = 0; i < n; ++i )
    for ( j = 0; j < i; ++j )
      if ( a[i][j] ) return 0;
  return 1;
}

int main() {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  int a[MAX][MAX], s;
  int n, i, j;

  printf( "Nhap bac ma tran: " );
  scanf( "%d", &n );
  for ( i = 0; i < n; ++i )
    for ( j = 0; j < n; ++j ) {
      printf( "a[%d][%d] = ", i, j );
      scanf( "%d", &a[i][j] );
    }

  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j <n; ++j )
      printf( "%5d", a[i][j] );

  for ( s = i = 0; i < n; ++i ) s += a[i][i];
  printf( "Trace = %d\n", s );

  if ( triMatrix( a, n ) ) {
    for ( s = 1, i = 0; i < n; ++i ) s *= a[i][i];
    printf( "det(A) = %d\n", s );
  }
  else printf( "A khong la ma tran tam giac tren\n" );
  return 0;
}
```

Hình bên là mô tả toán học của ma trận vuông cần thao tác. Với `i` là biến đếm của dòng và `j` là biến đếm của cột, các phần tử nằm trên đường chéo chính thỏa điều kiện `j = i` và các phần tử nằm trên đường chéo phụ thỏa điều kiện `j = n - i - 1`. Tam giác dưới của ma trận là miền thỏa điều kiện `j < i`.

Như vậy, các yêu cầu của bài tập có thể dễ dàng thực hiện:

- Kiểm tra ma trận tam giác dưới: ta kiểm tra các phần tử thuộc "miền" `j < i` xem có phần tử nào khác `0`, nếu có thì không phải là ma trận tam giác dưới.
- Tính `trace` và `det`: được tính dựa trên các phần tử thuộc đường chéo chính. Có thể duyệt các phần tử này bằng điều kiện `j = i`:

```
for ( s = i = 0; i < n; ++i )
  for ( j = 0; j < n; ++j )
    if ( j == i ) s += a[i][j];
```

Do `j = i` và `j` cùng miền xác định với `i` (miền `[0, n)`), nên chỉ cần một vòng lặp:

```
for ( s = i = 0; i < n; ++i ) s += a[i][i];
```

Bài 88: (trang 26)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 20
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
int isIdentity( int a[][MAX], int n ) {
  int i, j;
  for ( i = 0; i < n; ++i )
    for ( j = 0; j < n; ++j ) {
      if ( i != j && a[i][j] != 0 ) return 0;
      if ( i == j && a[i][j] != 1 ) return 0;
    }
  return 1;
}

int main() {
  int a[MAX][MAX];
  int n, i, j;

  printf( "Nhap bac ma tran: " );
  scanf( "%d", &n );

  srand( time( NULL ) );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%5d", a[i][j] = rand() % 201 - 100 );

  if ( isIdentity( a, n ) ) printf( "Ma tran dong nhat\n" );
  else {
    printf( "Ma tran tren khong dong nhat\n");
    for ( i = 0; i < n; ++i, putchar( '\n' ) )
      for ( j = 0; j < n; ++j )
        printf( "%5d", i == j );
  }
  return 0;
}
```

Bài 89: (trang 26)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 20
#define swap( a, b ) { int t = a; a = b; b = t; }

int main() {
  int a[MAX][MAX];
  int n, i, j;

  printf( "Nhap bac ma tran: " );
  scanf( "%d", &n );

  srand( time( NULL ) );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%5d", a[i][j] = rand() % 201 - 100 );

  for ( i = 0; i < n; ++i ) {
    int maxpos = 0;
    for ( j = 1; j < n; ++j )
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
      if ( a[i][j] > a[i][maxpos] ) maxpos = j;
    swap( a[i][i], a[i][maxpos] );
  }

  printf( "Ma tran sau khi sap xep:\n" );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%5d", a[i][j] );
  return 0;
}
```

Thao tác hoán chuyển phần tử có trị lớn nhất trong một dòng với phần tử nằm trên đường chéo chính thuộc dòng đó được thực hiện đơn giản như sau:

- Trên dòng `i`, tìm vị trí phần tử có trị lớn nhất, xem giải thuật bài 67 (trang 122). Giả sử đó là phần tử `a[i][maxpos]`.
- Hoán chuyển phần tử `a[i][maxpos]` với phần tử `a[i][i]` (phần tử này nằm trên đường chéo chính và thuộc dòng `i`).

Trong trường hợp gọi hàm, bạn nên tham khảo cách giải sau, thể hiện rõ tính linh hoạt của C khi xem mảng hai chiều như là mảng của mảng, tức mảng một chiều:

```
/* Hàm swapMax tìm phần tử có trị lớn nhất trong mảng
   một chiều b và hoán chuyển với phần tử b[p] */
void swapMax( int b[], int n, int p ) {
  int i, maxpos = 0;
  for ( i = 1; i < n; ++i )
    if ( b[i] > b[maxpos] ) maxpos = i;
  swap( b[p], b[maxpos] );
}

/* xem mỗi dòng như một mảng một chiều tên (a[i]), hoán chuyển
   phần tử có trị lớn nhất trong mảng (a[i]) với phần tử (a[i])[i] */
for ( i = 0; i < n; ++i )
  swapMax( a[i], n, i );
```

Bài 90: (trang 26)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 20

int main() {
  int a[MAX][MAX], k;
  int n, i, j;

  printf( "Nhap bac ma tran: " );
  scanf( "%d", &n );

  srand( time( NULL ) );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%5d", a[i][j] = rand() % 201 - 100 );

  printf( "Cac duong cheo song song cheo phu:\n" );
  for ( k = -n + 1; k < n; ++k, putchar( '\n' ) )
    for ( i = 0; i < n; ++i )
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
      for ( j = 0; j < n; ++j )
        if ( j == n - i - 1 + k )
          printf( "%5d", a[i][j] );
  return 0;
}
```

Mô tả toán học của bài tập này được trình bày trong hình dưới: các đường chéo song song đường chéo phụ phải thuộc họ các đường thẳng có phương trình tham số: `j = n - i - 1 + k` với tham số `k ∈ [-n + 1, n)`, trong đó `k = 0` chính là đường chéo phụ. Các phần tử `a[i][j]` nằm trên các đường chéo này dĩ nhiên phải thỏa điều kiện `i, j ∈ [0, n)`.

Như vậy, với mỗi `k` thuộc miền xác định ta tách lấy các phần tử `a[i][j]` thỏa các điều kiện trên (`j = n - i - 1 + k` và `i, j ∈ [0, n)`) thì các phần tử này sẽ nằm trên một đường chéo song song với đường chéo phụ, tương ứng với `k` đó.

Bài 91: (trang 27)

```
#include <stdio.h>
#include <stdlib.h>
#define _W 12
#define _H 8
#define _X 4
#define _Y 3

int p[4][6] = { { 0,1,0,1,0,0 },
                { 1,0,0,0,1,0 },
                { 1,0,0,0,1,0 },
                { 0,1,0,1,0,0 } };

int main() {
  int i, j;

  for ( i = 0; i < _H; ++i, putchar( '\n' ) )
    for ( j = 0; j < _W; ++j )
      printf( "%c", p[(i + _Y) % 4][(j + _X) % 6] ? '*' : ' ' );
  return 0;
}
```

Khai báo khởi tạo ma trận hai chiều như trên là tường minh, giúp hình dung được mảng hai chiều. Vì C xem mọi mảng là mảng một chiều nên có thể khai báo như sau, áp dụng cả với mảng có số chiều nhiều hơn 2:

```
int p[4][6] = { { 0,1,0,1,0,0 },
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
                { 1,0,0,0,1,0 },
                { 1,0,0,0,1,0 },
                { 0,1,0,1,0,0 } };
```

Hình trang sau mô tả trực quan yêu cầu của bài tập, ta có các nhận xét sau:

- Với ma trận `A`, tập chỉ số `i` ánh xạ vào đoạn `[0, 4)` (số dòng của ma trận `p`) và tập chỉ số `j` ánh xạ vào đoạn `[0, 6)` (số cột của ma trận `p`).

Nói cách khác, phần tử `A[i][j]` của ma trận `A` ánh xạ với phần tử `p[i % 4][j % 6]` của ma trận đơn vị `p`. Nếu `p[i % 4][j % 6]` chứa trị `1` thì tại `A[i][j]` vẽ ký tự `*`, ngược lại tại `A[i][j]` vẽ ký tự `space`.

- Ma trận `B` thực chất là phép dịch chuyển tịnh tiến của ma trận `A`, với độ dịch chuyển hai chiều là `(_Y, _X)`. Nghĩa là: phần tử `B[i][j]` của ma trận `B` tương ứng với phần tử `A[i + _Y][j + _X]` của ma trận `A`; và như vậy sẽ ánh xạ với phần tử `p[(i + _Y) % 4][(j + _X) % 6]` của ma trận đơn vị `p`.

Bài 92: (trang 27)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 20

int main() {
  int a[MAX][MAX], k, s, maxsum;
  int n, i, j, maxpos;

  printf( "Nhap bac ma tran: " );
  scanf( "%d", &n );

  srand( time( NULL ) );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%5d", a[i][j] = rand() % 201 - 100 );

  maxsum = -101;
  for ( k = -n + 1; k < n; ++k ) {
    for ( s = i = 0; i < n; ++i )
      for ( j = 0; j < n; ++j )
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
        if ( j == i + k ) s += a[i][j];
    if ( s > maxsum ) { maxsum = s; maxpos = k; }
  }

  printf( "Duong cheo co tong lon nhat:\n" );
  for ( i = 0; i < n; ++i )
    for ( j = 0; j < n; ++j )
      if ( j == i + maxpos ) printf( "%d ", a[i][j] );
  printf( ": %d\n", maxsum );
  return 0;
}
```

Đây là bài tập mở rộng của bài tập 90 (trang 152). Trong đó, ta làm việc với họ các đường thẳng song song với đường chéo chính, có phương trình tham số là: `j = i + k`.

Bài tập yêu cầu xác định tham số `k` ứng với đường chéo có tổng trị các phần tử lớn nhất. Sau khi xác định tham số `k` này (lưu trong `maxpos`) ta dễ dàng xuất các phần tử trên đường chéo tương ứng.

Chú ý cách chúng ta khởi tạo trị `maxsum` (`-101`), vì có đường chéo đầu tiên chỉ có một phần tử và phần tử này có trị lớn hơn `-101` nên trị `maxsum` chắc chắn sẽ được đặt lại sau khi tính tổng đường chéo đầu tiên này.

Bài 93: (trang 27)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 3

typedef struct {
  int array[MAX][MAX];
} array_inside;

array_inside mulMatrix( int a[][MAX], int b[][MAX] ) {
  array_inside temp = { { { { 0 } } } };
  int i, j, k;
  for ( i = 0; i < MAX; ++i )
    for ( j = 0; j < MAX; ++j )
      for ( k = 0; k < MAX; ++k )
        temp.array[i][j] += a[i][k] * b[k][j];
  return temp;
}

int main() {
  int a[MAX][MAX];
  array_inside c;
  int i, j, k;

  srand( time( NULL ) );
  for ( i = 0; i < MAX; ++i, putchar( '\n' ) )
    for ( j = 0; j < MAX; ++j )
      printf( "%5d", a[i][j] = c.array[i][j] = rand() % 10 );
  do {
    printf( "Nhap luy thua (k > 1): " );
    scanf( "%d", &k );
  } while ( k < 2 );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  for ( i = 2; i <= k; ++i )
     c = mulMatrix( a, c.array );

  for ( i = 0; i < MAX; ++i, putchar( '\n' ) )
    for ( j = 0; j < MAX; ++j )
      printf( "%5d", c.array[i][j] );
  return 0;
}
```

Nhắc lại, các chú ý khi khai báo mảng:

- Số các trị được khởi tạo phải bằng đúng với số phần tử khai báo trong mảng:

```
int a[5] = { 1, 2, 3, 4, 5 };
int b[2][3] = { { 1, 2, 3 }, { 4, 5, 6 } };
```

Số các trị được khởi tạo nếu ít hơn số phần tử khai báo của mảng, thì các phần tử còn lại của mảng sẽ được khởi tạo bằng `0`:

```
int a[5] = { 1, 2 };  /* a[2] đến a[4] chứa trị 0 */
/* b là ma trận tam giác dưới */
int b[3][3] = { { 1 }, { 4, 5 }, { 7, 8, 9 } };
```

- Nếu kích thước của mảng không được khai báo, thì số phần tử khởi tạo sẽ xác định kích thước của mảng.

```
int a[] = { 1, 2, 3, 4, 5 };
int b[][3] = { { 1 }, { 4 , 5 }, { 7, 8, 9 } };
```

Như vậy:

- Khai báo mảng `a` và khởi tạo tất cả các phần tử với trị `0`:

```
int a[MAX] = { 0 };
```

- Khai báo mảng 2 chiều `b` và khởi tạo tất cả các phần tử với trị `0`:

```
int b[MAX][MAX] = { { 0 } };
```

Khi thực hiện phép nhân ma trận trong hàm `mulMatrix()`, cần khởi tạo trước ma trận tạm `t` sao cho tất cả các phần tử đều có trị `0`, ta khai báo như trên.

Mặc dù khi khai báo `static` thì các phần tử của mảng cũng được khởi tạo bằng `0`, nhưng không được khai báo `static` với ma trận tạm `t` trong hàm `mulMatrix()`, vì nếu hàm này được gọi nhiều lần ma trận tạm `t` vẫn lưu giữ kết quả tính của lần trước.

Bạn cũng nên chú ý đến cách thiết kế hàm `mulMatrix()`:

- Cách thông thường, matrix `result` là tham số xuất, nhận kết quả nhân hai matrix `a` và `b`. Các tham số thường bố trí giống như toán tử gán, kết quả được gán sang trái.

```
void mulMatrix(result[][MAX], a[][MAX], b[][MAX]);
```

- Trả về con trỏ, bạn phải thực hiện cấp phát động trong hàm `mulMatrix()` và giải phóng con trỏ bên ngoài.
- Hàm `mulMatrix()` không thể trả về mảng hai chiều nhưng có thể trả về một structure bao bọc (wrapper) mảng hai chiều đó. Cách này được thực hiện ở trên.

Ma trận lũy thừa thường dùng để tính nhanh số Fibonacci:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} Fi(n+1) & Fi(n) \\ Fi(n) & Fi(n-1) \end{pmatrix}$$

```
int main() {
  int a[2][2] = { 1, 1, 1, 0 };
  array_inside c = { { 1, 1, 1, 0 } };
  int i, n;

  printf( "Nhap n (0 < n < 40): " );
  scanf( "%d", &n );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  printf( "Fibonacci( 1 ) = %d\n", c.array[0][1] );
  for ( i = 2; i <= n; ++i ) {
    c = mulMatrix( a, c.array );
    printf( "Fibonacci( %d ) = %d\n", i, c.array[0][1] );
  }
  return 0;
}
```

Bài 94: (trang 28)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#define MAX 20

void maxAbs( int a[][MAX], int n, int *maxr, int *maxc ) {
  int i, j;
  *maxr = *maxc = 0;
  for ( i = 0; i < n; ++i )
    for ( j = 0; j < n; ++j )
      if ( abs( a[i][j] ) > abs( a[*maxr][*maxc] ) )
        { *maxr = i; *maxc = j; }
}

int main() {
  int a[MAX][MAX], b[MAX][MAX];
  int n, i, j, maxr, maxc;

  printf( "Nhap bac ma tran (n > 1): " );
  scanf( "%d", &n );

  srand( time( NULL ) );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%5d", a[i][j] = rand() % 201 - 100 );

  maxAbs( a, n, &maxr, &maxc );
  for ( i = 0; i < n; ++i )
    for ( j = 0; j < n; ++j )
      b[i - ( i > maxr )][j - ( j > maxc )] = a[i][j];

  printf( "Ma tran B:\n" );
  for ( i = 0; i < n - 1; ++i, putchar( '\n' ) )
    for ( j = 0; j < n - 1; ++j )
      printf( "%5d", b[i][j] );
  return 0;
}
```

Nhắc lại, C hỗ trợ một tập đầy đủ các toán tử so sánh. Mỗi phép so sánh cho kết quả thành công hay thất bại. Kết quả định trị biểu thức so sánh bằng `1` nếu so sánh thành công, và bằng `0` nếu so sánh thất bại. Về mặt nguyên tắc, một trị khác `0` cũng có thể được dùng để chỉ phép so sánh thành công. Tuy nhiên C chỉ dùng `1` để chỉ so sánh thành công.

Bài tập: So sánh hai phân số `a/b` và `c/d` với `a`, `b`, `c`, `d` được nhập từ bàn phím.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
Nhap a b c d: 1 2 1 3 ↵
1/2 > 1/3
```

```
#include <stdio.h>

int comp( int a, int b ) {
  return ( a > b ) - ( a < b );
}

int main() {
  char op[] = { '<', '=', '>' };
  int a, b, c, d, z;

  printf( "Nhap a b c d: " );
  scanf( "%d%d%d%d", &a, &b, &c, &d );
  printf( "%d/%d %c %d/%d\n", a, b, op[comp( a * d, c * b ) + 1], c, d );
  return 0;
}
```

Hàm `compare()` so sánh `a` và `b`, trả về `-1`, `0` hoặc `1` nếu `a` nhỏ hơn `b`, bằng `b` hoặc lớn hơn `b`. Sau khi quy đồng mẫu số, việc so sánh hai phân số `a/b` và `c/d` chính là so sánh hai tử số `a * d` và `c * b`.

Một cách khác:

```
int comp( int a, int b ) {
  return ( a < b ) ? -1 : ( a > b );
}
```

Hình dưới trình bày cách loại bỏ dòng `maxr` và cột `maxc` của ma trận `a` khi sao chép nó sang ma trận `b` khác.

Cấu trúc điều kiện như sau:

```
if ( i > maxr ) b[i - 1][j] = a[i][j];                  /* dồn dòng */
else if ( j > maxc ) b[i][j - 1] = a[i][j];             /* dồn cột */
else /* dồn cả dòng lẫn cột */
  if ( ( i > maxr ) && ( j > maxc ) b[i - 1][j - 1] = a[i][j]
  else                              b[i][j] = a[i][j]; /* không thay đổi */
```

Từ kiến thức về thể hiện Đúng Sai của C khi định trị biểu thức so sánh, biểu thức phức tạp trên được viết gọn lại như sau:

```
b[i - (i > maxr)][j - (j > maxc)] = a[i][j];
```

Xóa bỏ dòng `r` và cột `c` của ma trận `a` cũng có thể thực hiện theo cách trên:

```
for ( i = 0; i < n; ++i )
  for ( j = 0; j < n; ++j )
    a[i - (i > r)][j - (j > c)] = a[i][j];
```

Xem thêm bài 64 (trang 119).

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Hàm `maxAbs()` dùng xác định vị trí của phần tử có trị tuyệt đối lớn nhất. Vì hàm này trả về đến 2 trị (dòng và cột), nên hai trị này được trả về dưới dạng tham số xuất và truyền bằng con trỏ.

Bài 95: (trang 28)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 20

int main() {
  int a[MAX][MAX], b[MAX][MAX];
  int n, i, j, k;

  printf( "Nhap bac ma tran: " );
  scanf( "%d", &n );

  srand( time( NULL ) );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%5d", a[i][j] = rand() % 201 - 100 );
  do {
    printf( "Nhap k: " );
    scanf( "%d", &k );
  } while ( k >= n );

  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; j++ )
      printf( "%5d", b[i][j] = ( i == k || j == k ) ? a[j][i] : a[i][j] );
  return 0;
}
```

Bài tập này chỉ đơn giản chuyển vị (transpose) dòng `k` với cột `k` của ma trận vuông với công thức chuyển vị quen thuộc `b[i][j] = a[j][i]`, nghĩa là dòng `k` từ `a` thành cột `k` của `b` và cột `k` từ `a` thành dòng `k` của `b`.

Chú ý khi bạn muốn chuyển vị trên cùng một ma trận vuông `a`:

```
#define swap(a, b) { int t = a; a = b; b = t; }
/* ... */
for ( i = 0; i < n; ++i )
  for ( j = 0; j < n; j++ )
    if ( i == k ) swap( a[i][j], a[j][i] );
```

nếu thêm điều kiện (`i == k || j == k`) như trên thì các vị trí chuyển vị hoán chuyển lặp lại làm cho mảng trở về ban đầu, ma trận `a` sẽ không thay đổi như mong muốn.

Bài 96: (trang 28)

```
#include <stdio.h>
#include <math.h>
#define MAX 20

int main() {
  double a[MAX][MAX];
  int n, i, j, k;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  printf( "Nhap bac ma tran: " );
  scanf( "%d", &n );

  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%10lf", a[i][j] = sin( i - 2 * j / M_PI ) );
  for ( k = i = 0; i < n; ++i )
    for ( j = 0; j < n; ++j )
      if ( a[i][j] >= 0 ) k++;
  printf( "Co %d phan tu khong am\n", k );
  return 0;
}
```

Bài 97: (trang 29)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 20

int minPos( int a[][MAX], int i, int j ) {
  int k;
  for ( k = j - 1; k <= j + 1; ++k )
    if ( ( a[i - 1][k] <= a[i][j] ) ||
         ( a[i + 1][k] <= a[i][j] ) ||
         ( a[i][j - 1] <= a[i][j] ) ||
         ( a[i][j + 1] <= a[i][j] ) ) return 0;
  return 1;
}

int main() {
  int  a[MAX][MAX];
  int n, m, i, j;

  printf( "Nhap n, m: " );
  scanf( "%d%d", &n, &m );

  srand( time( NULL ) );
  for ( i = 1; i <= n; ++i, putchar( '\n' ) )
    for ( j = 1; j <= m; ++j )
      printf( "%5d", a[i][j] = rand() % 201 - 100 );
  /* tạo biên giả cho ma trận */
  for ( j = 0; j <= m + 1; ++j )
    a[0][j] = a[n + 1][j] = 101;
  for ( i = 1; i <= n; ++i )
    a[i][0] = a[i][m + 1] = 101;

  printf( "Ma tran cuc tieu:\n" );
  for ( i = 1; i <= n; ++i, putchar( '\n' ) )
    for ( j = 1; j <= m; ++j )
      printf( "%5d", minPos( a, i, j ) );
  return 0;
}
```

Không khó xét cực tiểu một phần tử trong ma trận. Tuy nhiên, vấn đề ở đây là có khá nhiều trường hợp cần kiểm tra khi xét cực tiểu cho một phần tử:

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

- Nếu phần tử nằm ở góc ma trận: có 3 lân cận cần kiểm tra.
- Nếu phần tử nằm ở biên ma trận: có 5 lân cận cần kiểm tra.
- Nếu phần tử nằm ở giữa ma trận: có 8 lân cận cần kiểm tra.

Các lân cận của phần tử nằm ở góc và ở biên ma trận lại khác nhau tùy theo góc hoặc biên. Tổng cộng 9 trường hợp cần xét. Để đơn giản hóa quá trình xét cực tiểu, ta tạo một biên giả cho ma trận. Hình bên minh họa việc xét cực tiểu một phần tử nằm ở góc ma trận với sự tham gia của biên giả.

Như vậy:

- Các phần tử xét cực tiểu đều là *phần tử nằm ở giữa* ma trận và chỉ có một trường hợp: có 8 lân cận cần kiểm tra.
- Do các phần tử của biên giả chứa trị (`101`) lớn hơn tất cả các phần tử nên việc so sánh chúng với phần tử xét cực tiểu luôn đúng, không ảnh hưởng kết quả kiểm tra.

Khả năng xuất hiện một phần tử cực tiểu là thấp, nên hàm `minPos()` chủ yếu kiểm tra một phần tử *không phải* là cực tiểu trong vòng lặp. Biểu thức điều kiện được dùng là phủ định của biểu thức điều kiện bảo đảm một phần tử là cực tiểu, được suy ra từ định lý De Morgan.

Trị trả về của hàm `minPos()` được chuyển trực tiếp thành phần tử của ma trận cực tiểu nhị phân.

Bài 98: (trang 29)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 20

int main() {
  int A[MAX][MAX], C[MAX][MAX];
  int n, m, i, j ;

  printf( "Nhap n, m: " );
  scanf( "%d%d", &n, &m );

  srand( time( NULL ) );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < m; ++j )
      printf( "%5d", A[i][j] = rand() % 201 - 100 );

  printf( "Ma tran sau khi quay:\n" );
  for ( i = 0; i < m; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%5d", C[i][j] = A[n - 1 - j][i] );
  return 0;
}
```

Thông thường bạn tính nhẩm xem vị trí `C[i][j]` trong ma trận mới là kết quả chuyển đổi của vị trí `A[i][j]` nào của ma trận cũ, để thiết lập công thức chuyển đổi vị trí. Sau đó tiến hành hai vòng lặp lồng duyệt tất cả các phần tử, dùng công thức này, chuyển tất cả các vị trí từ ma trận cũ sang ma trận mới.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Cần phải có tư duy toán học ngay cả trong trường hợp bạn có thể tính nhẩm được, bởi vì công thức mà bạn tính nhẩm là kết quả của việc thử sai, của việc quy nạp chưa đầy đủ nên không chắc là chính xác. Quan sát hình dưới đây:

Phần trong khung là ma trận ta đã chuyển thành hệ tọa độ toán học để dễ tính toán, ta nhận thấy lúc này trục x là i và trục y là j. Phần trong khung cũng mô tả các bước ta quay và tịnh tiến ma trận để đáp ứng yêu cầu bài toán (chữ A để dễ nhận thấy ma trận quay).

Ma trận B là kết quả phép quay ma trận A một góc -90⁰ (theo chiều kim đồng hồ dấu -, ngược chiều kim đồng hồ dấu +), với tâm quay là gốc tọa độ:

$$\begin{pmatrix} xB \\ yB \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} xA \\ yA \end{pmatrix} \quad \alpha = -90^o$$

$$\begin{cases} xB = xA\cos\alpha - yA\sin\alpha \\ yB = xA\sin\alpha + yA\cos\alpha \end{cases} = \begin{cases} xB = yA \\ yB = -xA \end{cases}$$

Ma trận C là kết quả phép tịnh tiến ma trận B một đoạn dịch chuyển (n - 1) theo chiều tăng trục y:

$$\begin{pmatrix} xC \\ yC \end{pmatrix} = \begin{pmatrix} xB \\ yB + (n-1) \end{pmatrix}$$

$$\begin{cases} xC = xB \\ yC = yB + (n-1) \end{cases} = \begin{cases} xC = yA \\ yC = -xA + (n-1) \end{cases} = \begin{cases} xC = yA \\ yC = n - 1 - xA \end{cases} \Leftrightarrow \begin{cases} xA = n - 1 - yC \\ yA = xC \end{cases}$$

Nghĩa là giá trị tại tọa độ (xC, yC) của ma trận C là giá trị lấy từ tọa độ (xA = n - 1 - yC, yA = xC) của ma trận A:

C(xC, yC) = A(n - 1 - yC, xC)

Từ chú ý ở trên: "trục x là i và trục y là j": C[i][j] = A[n - 1 - j][i]

Đây chính là công thức cần tìm.

Bạn có thể tự hỏi: người ta đã tính nhẩm như thế nào để có công thức trên? Trước hết ta lấy mẫu và tạo kết quả đích bằng tay:

```
    1  2  3          7  4  1
A = 4  5  6      C = 8  5  2
    7  8  9          9  6  3
```

Dễ nhận thấy rằng hàng đã chuyển thành cột và cột đã chuyển thành hàng, ta nghĩ ngay đến công thức B[i][j] = A[j][i]

Nhưng đây là công thức quen thuộc của ma trận chuyển vị, kết quả khi chuyển vị như sau:

```
    1  4  7
B = 2  5  8
    3  6  9
```

So với kết quả đích, ta thấy C chỉ khác B ở thứ tự các cột: cột của C ngược lại so với cột của B, nghĩa là cột (j) của C là cột (n - 1 - j) của B:

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

`C[i][j] = B[i][n - 1 - j] = A[n - 1 - j][i]`

Bạn chắc nhận thấy ngay cách này nhanh nhưng không có cơ sở vững chắc và không phải trường hợp nào cũng tính nhẩm được.

Bài 99: (trang 29)

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

int** myAlloc( int n, int f ) {
  int i, j;
  int** t = ( int** )calloc( n, sizeof( int* ) );
  if ( !t ) return t;
  for ( i = 0; i < n; ++i ) {
    t[i] = ( int* ) calloc( n, sizeof( int ) );
    if ( f )
      for ( j = 0; j < n; ++j )
        t[i][j] = rand() % 21 - 10;
  }
  return t;
}

void printMatrix( int** a, int n, char* s ) {
  int i, j;
  printf( "%s:\n", s );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%5d", a[i][j] );
}

int** mulMatrix( int** a, int** b, int n ) {
  int** t;
  int i, j, k;
  if( ( t = myAlloc( n, 0 ) ) == NULL ) return t;
  for ( i = 0; i < n; ++i )
    for ( j = 0; j < n; ++j )
      for ( k = 0; k < n; ++k )
        t[i][j] += a[i][k] * b[k][j];
  return t;
}

int** addMatrix( int** a, int** b, int n ) {
  int** t;
  int i, j;
  if( ( t = myAlloc( n, 0 ) ) == NULL ) return t;
  for ( i = 0; i < n; ++i )
    for ( j = 0; j < n; ++j )
      t[i][j] = a[i][j] + b[i][j];
  return t;
}

void myFree( int** a, int n ) {
  int i;
  for ( i = 0; i < n; ++i ) free( a[i] );
  free( a );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
}

int main() {
  int **a, **b, **c, **d;
  int n;

  srand( time( NULL ) );
  printf( "Nhap bac ma tran: " );
  scanf( "%d", &n );

  a = myAlloc( n, 1 );
  b = myAlloc( n, 1 );

  if ( !a || !b )
    { printf( "Loi cap phat\n" ); return 1; }

  printMatrix( a, n, "Ma tran A" );
  printMatrix( b, n, "Ma tran B" );

  c = addMatrix( a, b, n );
  if ( c ) printMatrix( c, n, "Ma tran tong" );
  else { printf( "Loi cap phat\n" ); return 1; }

  d = mulMatrix( a, b, n );
  if ( d ) printMatrix( d, n, "Ma tran tich" );
  else { printf("Loi cap phat\n"); return 1; }

  myFree( a, n ); myFree( b, n );
  myFree( c, n ); myFree( d, n );
  return 0;
}
```

Bài tập yêu cầu bạn biết cách cấp phát vùng nhớ động trên heap cho mảng 2 chiều trong C. Giả sử ta cần cấp phát động một mảng hai chiều có kích thước `n x m`, có hai cách thực hiện điều này:

- Cách 1:

Đầu tiên ta cấp phát một mảng chứa `n` con trỏ kiểu `int*`, quản lý bởi con trỏ `t` (kiểu `int**`). Tiếp theo, với mỗi con trỏ trong mảng con trỏ `t`, cấp phát một mảng chứa `m` phần tử kiểu `int`. Do cấp phát nhiều lần nên với cách này vùng nhớ được cấp phát không liên tục.

```
Cấp phát:
t = ( int** )calloc( n, sizeof( int* ) );
for ( i = 0; i < n; ++i )
   t[i] = ( int* )calloc( m, sizeof( int ) );

Giải phóng:
for ( i = 0; i < n; ++i )
  free( t[i] );
free( t );
```

- Cách 2:

Đầu tiên ta cấp phát một mảng chứa `n` con trỏ kiểu `int*`, quản lý bởi con trỏ `t` (kiểu `int**`). Tiếp theo, cấp phát một vùng nhớ đủ cho `n x m` phần tử của mảng hai chiều, quản lý bởi con trỏ đầu tiên `t[0]` của mảng con trỏ `t`. Cuối cùng, "chia" đều vùng nhớ này cho các con trỏ còn lại trong mảng con trỏ `t`.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Vùng nhớ cấp phát bởi cách này liên tục hơn, gọi là chế độ phẳng (flattened mode), giống cách C cấp phát tĩnh cho mảng hai chiều.

Cấp phát:
```
t = ( int** )calloc( n, sizeof( int* ) );
t[0] = ( int* )calloc( n * m, sizeof( int ) );
for ( i = 0; i < n; ++i )
  t[i] = t[0] + i * m;
```

Giải phóng:
```
free( t[0] );
free( t );
```

Sau khi cấp phát các ma trận được sử dụng bình thường giống như trường hợp các ma trận cấp phát tĩnh.

Vì các ma trận tham gia vào các phép tính đều là ma trận vuông và cùng cấp nên những yêu cầu cho các phép tính trên ma trận đều thỏa.

Cần chú ý hàm `myAlloc()`, khởi tạo cho các ma trận vừa cấp phát khác nhau tùy theo tham số `f`: nếu bằng `0` thì các phần tử của ma trận mới cấp phát sẽ được khởi tạo với trị `0`; ngược lại, các phần tử của ma trận mới cấp phát sẽ được khởi tạo với trị ngẫu nhiên nằm trong đoạn `[-10, 10]`.

Bài 100: (trang 30)

```
#include <stdio.h>
#define MAX 20

int main() {
  int a[MAX][MAX];
  int n, m, i, j, s;

  printf( "Nhap n, m: " );
  scanf( "%d%d", &n, &m );

  for ( i = 0; i < n; ++i )
    for ( j = 0; j < m; ++j ) {
      printf( "a[%d][%d] = ", i, j );
      scanf( "%d", &a[i][j] );
    }
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < m; ++j )
      printf( "%5d", a[i][j] );

  for ( s = i = 0; i < n; ++i )
    for ( j = 0; j < m; ++j )
      s += ( !a[i][j] );

  if ( 2 * s > n * m ) printf( "Ma tran thua\n" );
  else                 printf( "Ma tran khong thua\n" );
  return 0;
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Chú ý cách đếm số phần tử `s` có trị `0` của mảng, bằng cách dùng biểu thức điều kiện. Hai lần trị `s` này sẽ được so sánh với số phần tử `n * m` của ma trận. Không nên so sánh (`s > n * m / 2`) để tránh phép chia nguyên.

Bài 101: (trang 30)

```
#include <stdio.h>
#define MAX 20

int isMagic( int a[][MAX], int n ) {
  int i, j, t, s1, s2;

  for ( s1 = s2 = i = 0; i < n; ++i ) {
    s1 += a[i][i];
    s2 += a[i][n - i - 1];
  }
  if ( s1 != s2 ) return 0;

  t = s1;
  for ( i = 0; i < n; ++i ) {
    for ( s1 = s2 = j = 0; j < n; ++j ) {
      s1 += a[i][j];
      s2 += a[j][i];
    }
    if ( s1 != t || s2 != t ) return 0;
  }
  return 1;
}

int main() {
  int  a[MAX][MAX];
  int n, i, j, sr, sc;

  printf( "Nhap bac ma tran: " );
  scanf( "%d", &n );

  for ( i = 0; i < n; ++i )
    for ( j = 0; j < n; ++j ) {
      printf( "a[%d][%d] = ", i, j );
      scanf( "%d", &a[i][j] );
    }

  for ( i = 0; i < n; ++i ) {
    for ( sr = sc = j = 0; j < n; ++j ) {
      sr += a[i][j];
      sc += a[j][i];
      printf( "%5d", a[i][j] );
    }
    printf( "   Tong hang %d:%4d", i, sr );
    printf( "   Tong cot %d:%4d\n", i, sc );
  }
  if ( isMagic( a, n ) ) printf( "Ma phuong\n" );
  else printf( "Khong phai ma phuong\n" );
  return 0;
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Khi tính tổng trị các phần tử của mỗi dòng, để thực hiện ý tưởng "với mỗi dòng", bạn dùng vòng `for` đếm dòng (biến đếm `i`) bên ngoài. Với mỗi dòng `i` bạn dùng vòng `for` đếm cột (biến đếm `j`) bên trong để cộng dồn các phần tử trên dòng đó:

```
for ( i = 0; i < n; ++i )
  for ( sr = j = 0; j < n; ++j )
    sr += a[i][j];
```

Khi tính tổng các cột, khi phân tích ta nhận thấy vai trò "dòng" và "cột" đổi chỗ nhau: "với mỗi dòng ta cộng dồn cột" thay bằng "với mỗi cột ta cộng dồn dòng". Vì vậy, thật đơn giản, bạn chỉ cần đổi chỗ các vòng lặp `i` và `j` tương ứng.

```
for ( j = 0; j < n; ++j )
  for ( sc = i = 0; i < n; ++i )
    sc += a[i][j];
```

Với ma trận vuông hai vòng lặp `i` và `j` có số lần lặp như nhau nên có thể viết:

```
for ( i = 0; i < n; ++i )
  for ( sc = j = 0; j < n; ++j )
    sc += a[j][i];
```

Như vậy, tổng trị phần tử các dòng và các cột được tính gọn hơn như sau:

```
for ( i = 0; i < n; ++i ) {
  for ( sr = sc = j = 0; j < n; ++j ) {
    sr += a[i][j];
    sc += a[j][i];
  }
```

Tổng trị các phần tử thuộc đường chéo chính và đường chéo phụ được tính như bài 87 (trang 149), chung trong một vòng lặp:

```
for ( s1 = s2 = i = 0; i < n; ++i ) {
  s1 += a[i][i];
  s2 += a[i][n-i-1];
}
```

Bài 102: (trang 30)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 20

int main() {
  int a[MAX][MAX], b[MAX][MAX];
  int n, m, i, j;

  printf( "Nhap n, m: " );
  scanf( "%d%d", &n, &m );

  srand( time( NULL ) );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < m; ++j )
      printf( "%5d", a[i][j] = rand() % 201 - 100 );

  printf( "Ma tran chuyen vi:\n" );
  for ( i = 0; i < m; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%5d", b[i][j] = a[j][i] );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  return 0;
}
```

Nếu ma trận là ma trận vuông cấp n và chuyển vị diễn ra chỉ trong ma trận đó, nghĩa là không tạo ma trận mới, thì ta chỉ cần chuyển vị tam giác dưới hoặc tam giác trên của ma trận.

Khi duyệt ma trận ta chỉ cần duyệt đủ "miền" cần hoán chuyển là tam giác dưới đường chéo chính, vì nếu duyệt cả ma trận thì khi hoán chuyển tam giác còn lại (trên đường chéo chính) bạn đã hoán chuyển một lần nữa các cặp đã hoán chuyển và như vậy ma trận trở về như ban đầu.

```
#define swap(a, b) { int t = a; a = b; b = t; }
/* ... */
for ( i = 0; i < n; ++i, putchar( '\n' ) )
  for ( j = 0; j < i; ++j )
    swap( a[i][j], a[j][i] );
```

Chú ý cặp phần tử đối xứng với nhau qua đường chéo chính là: (`a[i][j], a[j][i]`), một thuộc tam giác dưới và một thuộc tam giác trên.

Bài 103: (trang 31)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 20

int main() {
  int a[MAX][MAX];
  int sumd1[2 * MAX], sumd2[2 * MAX];
  int n, i, j, k;

  printf( "Nhap bac ma tran: " );
  scanf( "%d", &n );

  srand( time( NULL ) );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%5d", a[i][j] = rand() % 21 - 10 );

  for ( k = -n + 1; k < n; ++k ) {
    int s1, s2;
    s1 = s2 = 0;
    for ( i = 0; i < n; ++i )
      for ( j = 0; j < n; ++j ) {
        if ( j == i + k ) s1 += a[i][j];
        if ( j == n - i - 1 + k ) s2 += a[i][j];
      }
    sumd1[k + n - 1] = s1;
    sumd2[k + n - 1] = s2;
  }

  printf( "Ma tran B:\n" );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%5d", sumd1[j - i + n - 1] + sumd2[j + i] - a[i][j] );
  return 0;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
}
```

Bạn cần các kiến thức về họ đường chéo song song đường chéo phụ (bài 90, trang 152) và họ đường chéo song song đường chéo chính (bài 92, trang 154) để giải bài tập này.

Cách giải được trình bày trong hình bên:

Với phần tử `a[i][j]` bất kỳ:

- Tổng trị `s1` của các phần tử trên đường chéo song song đường chéo chính đi qua `a[i][j]` được tính và đặt vào mảng `sumd1`. Đường chéo này có phương trình tham số là `j = i + k1` (1), vì `k1 ∈ [-n + 1, n)` nên chỉ số của nó trong mảng `sumd1` phải được tịnh tiến thành: `k1 + n - 1` (2). Từ (1) và (2) ta có chỉ số của tổng trị `s1` trong mảng `sumd1` theo `i, j` là: `j - i + n - 1`.
- Tính toán tương tự, ta có tổng trị `s2` của các phần tử trên đường chéo song song đường chéo phụ đi qua `a[i][j]` được tính và đặt vào mảng `sumd2` tại phần tử có chỉ số theo `i, j` là: `j + i`.

Sau khi tính xong hai mảng `sumd1` và `sumd2`, ta dễ dàng tính được tổng trị các phần tử trên hai đường chéo (song song đường chéo chính và đường chéo phụ) đi qua phần tử `a[i][j]`:

```
b[i][j] = s1 + s2 = sumd1[j - i + n - 1] + sumd2[j + i] - a[i][j]
```

phần tử `a[i][j]` có mặt cả trong hai tổng `s1` và `s2` nên cần loại bớt một phần tử.

Tuy nhiên, do đặc điểm của phần tử nằm trên hai đường chéo đi qua một phần tử chỉ định, ta có cách giải ngắn gọn hơn khi xét với điều kiện đơn giản như sau:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#define MAX 20

int main() {
  int a[MAX][MAX];
  int n, i, j, k, l, s;

  printf( "Nhap bac ma tran: " );
  scanf( "%d", &n );

  srand( time( NULL ) );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%5d", a[i][j] = rand() % 21 - 10 );

  printf( "Ma tran B:\n" );
  for ( k = 0; k < n; ++k, putchar( '\n' ) )
    for ( l = 0; l < n; ++l ) {
      for ( s = i = 0; i < n; ++i )
        for ( j = 0; j < n; ++j )
          if ( abs(i - k) == abs(j - l) ) s += a[i][j];
      printf( "%5d", s );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    }
  return 0;
}
```

Bài 104: (trang 31)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#define MAX 20

int main() {
  int a[MAX][MAX];
  int n, i, j, k, l, max;

  printf( "Nhap bac ma tran: " );
  scanf( "%d", &n );

  srand( time( NULL ) );
  for ( i = 0; i < n ; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%5d", a[i][j] = rand() % 21 - 10 );

  printf( "Ma tran B:\n" );
  for ( k = 0; k < n; ++k, putchar( '\n' ) )
    for ( l = 0; l < n; ++l ) {
      max = -11;
      for ( i = 0; i < n; ++i )
        for ( j = 0; j < n; ++j )
          if ( abs(i - k) == abs(j - l) && a[i][j] > max )
            max = a[i][j];
      printf( "%5d", max );
    }
  return 0;
}
```

Bài tập này được thực hiện giống bài 103 (trang 170), chú ý trị `max` khởi tạo nhỏ hơn biên trái của đoạn trị được chọn cho các phần tử của ma trận nên chắc chắn `max` sẽ bị thay thế.

Bài 105: (trang 31)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 20

int main() {
  int a[MAX][MAX];
  int sumR[MAX] = { 0 };
  int sumC[MAX] = { 0 };
  int n, i, j, s;

  printf( "Nhap bac ma tran: " );
  scanf( "%d", &n );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  srand( time( NULL ) );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%5d", a[i][j] = rand() % 21 - 10 );

  for (s = i = 0; i < n; s += sumR[i++] )
    for ( j = 0; j < n; ++j ) {
      sumR[i] += a[i][j];
      sumC[i] += a[j][i];
    }

  printf( "Ma tran B:\n" );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%5d", s - ( sumR[i] + sumC[j] - a[i][j] ) );
  return 0;
}
```

Việc tính tổng trị các phần tử của mỗi dòng và mỗi cột được giải thích trong bài 101 (trang 166): Tổng trị các phần tử của mỗi dòng lưu trong mảng `sumR`, chỉ số của mảng là chỉ số của dòng tương ứng. Tổng trị các phần tử của mỗi cột lưu trong mảng `sumC`, chỉ số của mảng là chỉ số của cột tương ứng. Các phần tử của hai mảng này đều được khởi tạo bằng `0`.

Như vậy, tổng trị các phần tử nằm trên dòng `i`, cột `j`, đi qua phần tử `a[i][j]` là:

`sumR[i] + sumC[j] - a[i][j]`

Phần tử `a[i][j]` có mặt cả trong hai tổng `sumR[i]` và `sumC[j]` nên cần loại bớt một phần tử.

Sau mỗi vòng lặp `j`, tức với từng (dòng) `i`,ta nhân tiện tính tổng trị `s` của các phần tử trong ma trận, thực chất là tổng trị các phần tử của mảng `sumR`.

Suy ra tổng trị các phần tử *không* thuộc dòng `i` cột `j` của ma trận `A`:

`b[i][j] = s - ( sumR[i] + sumC[j] - a[i][j] )`

Bài 106: (trang 32)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 20

int main() {
  int a[MAX][MAX];
  int maxc[MAX], minc[MAX];
  int n, i, j, pmax, pmin;

  printf( "Nhap bac ma tran: " );
  scanf( "%d", &n );
  srand( time( NULL ) );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%5d", a[i][j] = rand() % 21 - 10 );

  /* mảng maxc: chứa chỉ số dòng của dòng có phần tử 'max cột'
     mảng minc: chứa chỉ số dòng của dòng có phần tử 'min cột'
     Chỉ số của mảng là cột. Nghĩa là, cột j có phần tử 'max cột' tại
     dòng maxc[j] và phần tử 'min cột' tại dòng minc[j] */
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  for ( j = 0; j < n; ++j ) {
    pmax = pmin = 0;
    for ( i = 1; i < n; ++i ) {
      if ( a[i][j] > a[pmax][j] ) pmax = i;
      if ( a[i][j] < a[pmin][j] ) pmin = i;
    }
    maxc[j] = pmax;
    minc[j] = pmin;
  }
  /* xác định cột có phần tử 'max dòng' và cột có phần tử 'min dòng' */
  for ( i = 0; i < n; ++i ) {
    pmax = pmin = 0;
    for ( j = 1; j < n; ++j ) {
      if ( a[i][j] > a[i][pmax] ) pmax = j;
      if ( a[i][j] < a[i][pmin] ) pmin = j;
    }
    /* pmin là cột có phần tử 'min dòng' của dòng i, phần tử 'max cột'
       của cột pmin có chỉ số dòng là maxc[pmin], ta so sánh xem
       chỉ số dòng này có phải là dòng i hay không */
    if ( i == maxc[pmin] )
      printf( "MIN dong MAX cot: a[%d][%d] = %d\n", i, pmin, a[i][pmin] );
    /* pmax là cột có phần tử 'max dòng' của dòng i, phần tử 'min cột'
       của cột pmax có chỉ số dòng là minc[pmax], ta so sánh xem
       chỉ số dòng này có phải là dòng i hay không */
    if  ( i == minc[pmax] )
      printf( "MAX dong MIN cot: a[%d][%d] = %d\n", i, pmax, a[i][pmax] );
  }
  return 0;
}
```

Xem chú giải chi tiết trong bài giải.

Bài 107: (trang 32)

```
#include <stdio.h>

int main() {
  int a[5][5];
  int n, i, j, v;

  n = 5;
  v = 1;
  for ( i = 0; i < n; ++i )
    for ( j = 0; j < n; ++j )
      if ( i % 2 ) a[i][n - j - 1] = v++;
      else         a[i][j] = v++;
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%3d", a[i][j] );
  return 0;
}
```

Bài tập này thực hiện đơn giản, các dòng lần lượt theo thứ tự từ trên xuống dưới sẽ được xử lý như sau:

- Với mỗi dòng chẵn: trị v tăng dần sẽ được gán vào từng phần tử của các cột tính từ trái sang phải một cách bình thường.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

- Với mỗi dòng lẻ: trị `v` tăng dần sẽ được gán vào từng phần tử của các cột tính từ phải sang trái, đổi chiều gán bằng cách áp dụng công thức cặp phần tử đối xứng nhau trong mảng một chiều (`b[i], b[n - i - 1]`).

Bài 108: (trang 33)

```
#include <stdio.h>
#define MAX 20
#define AK a[i][j] = ++k; if ( k == n * n ) goto RET;

int main() {
  int a[MAX][MAX];
  int i, j, k, n;

  printf( "Nhap bac ma tran (n < 20): " );
  scanf( "%d", &n );

  i = j = k = 0;
  while( 1 ) {
    do { AK; j++; } while ( j < n - i - 1 );
    do { AK; i++; } while ( j > i );
    do { AK; j--; } while ( j > n - i - 1 );
    do { AK; i--; } while ( j < i - 1 );
  }
RET:
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%3d", a[i][j] );
  return 0;
}
```

Bài tập có vẻ phức tạp nhưng được thực hiện dễ dàng. Hình dưới là mô tả toán học của bài tập trên:

Chú ý trong một vòng xoắn ốc, các cạnh xoắn ốc không phải bị giới hạn bởi các biên của ma trận như cách nghĩ thông thường, mà bị giới hạn bởi đường chéo chính `j = i`, đường chéo phụ `j = n - i - 1` và đường chéo song song đường chéo chính `j = i - 1`. Các đường chéo này chia mặt phẳng thành nhiều miền. Tùy điều kiện mỗi miền ta xác định được yêu cầu cần thực hiện trong mỗi miền như: tăng `j` để đi dần sang phải, giảm `i` để đi từ dưới lên trên, … Ví dụ:

```
do {
  a[i][j] = ++k;     /* trị k tăng dần được gán vào từng phần tử */
  if ( k == n * n )
    goto RET;        /* nếu đã gán đầy ma trận thì thoát */
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  j--;                /* trong miền j < n-i-1, giảm j để đi từ phải sang trái */
} while ( j > n - i - 1 );
```

Macro AK, dùng gán lần lượt các trị k tăng dần vào các phần tử của ma trận và kiểm tra đã gán xong, giúp code viết ngắn gọn hơn. Bạn cần học tập sử dụng thông thạo macro, lập trình viên C sử dụng rất nhiều macro tự viết hoặc có sẵn.
Phát biểu goto giúp thoát nhanh ra khỏi hai vòng lặp: vòng lặp while vĩnh viễn bên ngoài và vòng lặp do while dùng gán trị bên trong.
Nếu tạo ma trận trong hàm, thay lệnh goto bằng return;
Nếu ma trận không vuông, tham khảo bài tập 111 (trang 176).

Bài 109: (trang 33)

```
#include <stdio.h>

int main() {
  int a[5][5];
  int n, i, j, k, v;

  n = 5;
  v = 1;
  for ( k = -n + 1; k < n; ++k )
    for ( i = 0; i < n; ++i )
      for ( j = 0; j < n; ++j )
        if ( j == n - i - 1 + k )
          if ( k % 2 ) a[j][i] = v++;
          else a[i][j] = v++;

  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < n; ++j )
      printf( "%3d", a[i][j] );
  return 0;
}
```

Bài tập có vẻ phức tạp này được thực hiện dễ dàng nếu bạn đã có kiến thức về họ đường chéo song song đường chéo phụ j = n - i - 1 + k (bài 90, trang 152) và kiến thức về ma trận chuyển vị (bài 102, trang 167).
Ta lần lượt gán các trị k tăng dần vào các phần tử nằm trên các đường chéo song song với đường chéo phụ, khi sang đường chéo khác (nghĩa là khi k thay đổi, kiểm tra bằng cách xét k chẵn hoặc lẻ), ta đổi chiều thứ tự các phần tử cần gán trị. Nhận xét thấy rằng các phần tử nằm trên một đường chéo song song đường chéo phụ tạo thành các cặp phần tử đối xứng nhau qua đường chéo chính, ta áp dụng công thức chuyển vị ma trận để tạo nên sự đổi chiều này.

Bài 110: (trang 33)

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

int** myAlloc( int n, int m ) {
  int i, j;
  int** t = ( int ** )calloc( n, sizeof( int* ) );
  if ( !t ) return t;
  t[0] = ( int * )calloc( n * m, sizeof( int ) );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  if ( !t[0] ) return t;
  srand( time( NULL ) );
  for ( i = 0; i < n; ++i ) {
    t[i] = t[0] + i * m;
    for ( j = 0; j < m; ++j )
      t[i][j] = rand() % 21 - 10;
  }
  return t;
}

void myFree( int** a, int n ) {
  free( a[0] );
  free( a );
}

int sumNeg( int *a, int size ) {
  int i, s;
  for ( s = i = 0; i < size; ++i )
    if ( a[i] < 0 ) s++;
  return s;
}

int main() {
  int **a;
  int n, m, i, j, k;

  printf( "Nhap n, m: " );
  scanf( "%d%d", &n, &m );

  a = myAlloc( n, m );
  if ( !a )
    { printf( "Loi cap phat\n" ); return 1; }

  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < m; ++j )
      printf( "%5d", a[i][j] );
  do {
    printf( "Nhap k: " );
    scanf( "%d", &k );
  } while ( k < 0 || k > n );
  printf( "Dong %d co %d so am\n", k, sumNeg( a[k], m ) );
  myFree( a, n );
  return 0;
}
```

Cấp phát động cho mảng hai chiều đã được trình bày chi tiết trong bài 99 (trang 163), bài tập này hiện thực cách cấp phát thứ hai (flattened mode).

- Do tính linh hoạt của C khi xử lý mảng, hàm áp dụng cho mảng có thể dùng áp dụng dễ dàng cho một dòng của ma trận:

```
sumNeg( b, n );        /* b là tên mảng một chiều b */
sumNeg( a[k], m );     /* a[k] là "tên" dòng k của ma trận a */
```

Ở đây `a[k]` được xem như "tên" của một mảng. Lập trình viên C có tư duy gần với kiến trúc bên dưới hơn. Họ không phân biệt trị được truyền là tên mảng `b` hay tên dòng thứ `k` của ma trận `a[k]`, mà chỉ xem đó là các địa chỉ bắt đầu của một vùng nhớ liên tục:

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
sumNeg( b, n );        /* b là địa chỉ bắt đầu mảng b */
sumNeg( a + k, m );    /* a+k là địa chỉ bắt đầu dòng a[k] của ma trận a */
```

Tham khảo thêm bài 89 (trang 151).

Bài 111: (trang 34)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define MAX 20

void rShiftkStep( int b[], int n, int m, int k ) {
  int i, j;

  for ( i = 0; i < k; ++i ) {
    int t = b[n * m - 1];
    for ( j = n * m - 1; j > 0; --j )
      b[j] = b[j - 1];
    b[0] = t;
  }
}

void two2one( int b[], int a[][MAX], int n, int m ) {
  int i, j, v, direction, k;
  int size = n * m;

  v = direction = 0;
  i = j = 0;
  do {
    switch ( direction ) {
      case 0:
        for ( k = 0; k < m; ++k ) b[v++] = a[i][j+k];
        j += m - 1; break;
      case 1:
        for ( k = 1; k < n; ++k ) b[v++] = a[i+k][j];
        i += n - 1; break;
      case 2:
        for ( k = 1; k < m; ++k ) b[v++] = a[i][j-k];
        j -= m - 1; break;
      case 3:
        for ( k = 1; k < n-1; ++k ) b[v++] = a[i-k][j];
        i -= n - 2;
        n -= 2;
        m -= 2;
        j++;
    }
    direction = ( direction + 1 ) % 4;
  } while ( v < size );
}

void one2two( int a[][MAX], int b[], int n, int m ) {
  int i, j, v, direction, k;
  int size = n * m;

  v = direction = 0;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  i = j = 0;
  do {
    switch ( direction ) {
      case 0:
        for ( k = 0; k < m; ++k ) a[i][j+k] = b[v++];
        j += m - 1; break;
      case 1:
        for ( k = 1; k < n; ++k ) a[i+k][j] = b[v++];
        i += n - 1; break;
      case 2:
        for ( k = 1; k < m; ++k ) a[i][j-k] = b[v++];
        j -= m - 1; break;
      case 3:
        for ( k = 1; k < n-1; ++k ) a[i-k][j] = b[v++];
        i -= n - 2;
        n -= 2;
        m -= 2;
        j++;
    }
    direction = ( direction + 1 ) % 4;
  } while ( v < size );
}

int main() {
  int a[MAX][MAX];
  int b[MAX*MAX];
  int n, m, i, j, k;

  printf( "Nhap dong, cot: ");
  scanf( "%d%d", &n, &m );

  srand( time( NULL ) );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < m; ++j )
      printf( "%5d", a[i][j] = rand() % 21 - 10 );

  two2one( b, a, n, m );    /* chuyển từ mảng 2 chiều sang 1 chiều */

  printf( "Nhap buoc dich: " );
  scanf( "%d", &k );
  if ( k > n * m ) k = 0;
  rShiftkStep( b, n, m, k );    /* dịch trên mảng một chiều */

  one2two( a, b, n, m );    /* chuyển từ mảng 1 chiều trở lại 2 chiều */

  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < m; ++j )
      printf( "%5d", a[i][j] );
  return 0;
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Khác hẳn bài 108 (trang 173), bài tập trên không dùng ma trận vuông nên chúng ta phải dùng các cạnh ma trận để giới hạn việc gán vào các phần tử trong từng chu kỳ của vòng xoắn ốc. Hình bên trình bày những tính toán cụ thể:

- Trong cạnh `0` của vòng xoắn, gán cho `m-1` phần tử trong một vòng lặp, sau đó tăng `j` đúng `m-1`. Tương tự với các cạnh `1` (gán `n-1` phần tử, tăng `i` đúng `n-1`), cạnh `2` (gán `m-1` phần tử, giảm `j` đúng `m-1`) và cạnh `3` (gán `n-2` phần tử, giảm `i` đúng `n-2`).
- Muốn chuyển sang chu kỳ vòng xoắn thứ hai, phải tăng `j` (`j++`), sau đó hiệu chỉnh lại kích thước các cạnh vòng xoắn: `m` và `n` đều giảm `2` phần tử.

Dùng những tính toán trên để viết hai hàm:
- `two2one()`: lần lượt lấy các phần tử theo thứ tự vòng xoắn ốc trong mảng hai chiều chuyển vào mảng một chiều.
- `one2two()`: thao tác ngược lại, lần lượt chuyển các phần tử của mảng một chiều tương ứng vào mảng hai chiều theo thứ tự vòng xoắn ốc.

Hai hàm này viết như nhau, chỉ khác ở vai trò gán phần tử.

Trước lúc chuyển mảng một chiều trở lại mảng hai chiều theo thứ tự vòng xoắn ốc ta dùng hàm `rShiftkStep()` dịch phải mảng một chiều `k` vị trí. Hàm này đã được giải thích trong bài tập 72 (trang 129).

Cách giải trên rất tường minh và dễ hiểu nhưng không gọn. Tham khảo cách giải cuối của bài 85 (trang 146) ta đặt vấn đề: có thể tính công thức ánh xạ chỉ số giữa mảng hai chiều và mảng một chiều ở trên được không? Nếu tính được thì ta có thể áp dụng công thức ánh xạ này trong hàm `rShiftkStep()` và không cần mảng một chiều trung gian. Song việc tính công thức ánh xạ chỉ số này rõ ràng phức tạp.

Tuy vậy, nhờ nhận xét này ta có bài giải gọn và đáng học tập hơn:
- Thay vì dùng mảng một chiều lưu *trị* phần tử tương ứng theo chiều xoắn ốc trên mảng hai chiều, ta dùng mảng một chiều lưu *chỉ số* phần tử tương ứng theo chiều xoắn ốc trên mảng hai chiều. Nói cách khác, ta dùng mảng một chiều không như một đối tượng trung gian, mà như một *cấu trúc dữ liệu* đặc biệt lưu thông tin truy xuất mảng hai chiều. Thông tin này, chính là thông tin ánh xạ chỉ số, sẽ được dùng trong hàm `rShiftkStep()`.
- Một phần tử của mảng một chiều lưu đến hai chỉ số `i` và `j` cho một phần tử của mảng hai chiều nên ta dùng công thức ánh xạ để lưu chung `i` và `j`: `i * size + j` (`size = n * m` là kích thước mảng hai chiều). Có thể dùng mảng các structure ở đây nhưng chúng ta chưa thảo luận đến các bài tập về structure.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define MAX 20

void rShiftkStep( int a[][MAX], int b[], int n, int m, int k ) {
  int i, j, t;
  int size = n * m;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  for ( i = 0; i < k; ++i ) {
    t = a[b[n * m - 1]/size][b[n * m - 1]%size];
    for ( j = n * m - 1; j > 0; --j )
      a[b[j]/size][b[j]%size] = a[b[j-1]/size][b[j-1]%size];
    a[0][0] = t;
  }
}

/* thay vì lưu trị, lưu chỉ số của phần tử tương ứng theo chiều xoắn ốc */
void createIndex( int b[], int a[][MAX], int n, int m ) {
  int i, j, v, direction, k;
  int size = n * m;

  v = direction = 0;
  i = j = 0;
  do {
    switch ( direction ) {
      case 0:
        for ( k = 0; k < m; ++k ) b[v++] = i*size + (j+k);
        j += m - 1; break;
      case 1:
        for ( k = 1; k < n; ++k ) b[v++] = (i+k)*size + j;
        i += n - 1; break;
      case 2:
        for ( k = 1; k < m; ++k ) b[v++] = i*size + (j-k);
        j -= m - 1; break;
      case 3:
        for ( k = 1; k < n-1; ++k ) b[v++] = (i-k)*size + j;
        i -= n - 2;
        n -= 2;
        m -= 2;
        j++;
    }
    direction = ( direction + 1 ) % 4;
  } while ( v < size );
}

int main() {
  int a[MAX][MAX];
  int b[MAX*MAX];                /* b là mảng ánh xạ chỉ số */
  int n, m, i, j, k;

  printf( "Nhap dong, cot: ");
  scanf( "%d%d", &n, &m );

  srand( time( NULL ) );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < m; ++j )
      printf( "%5d", a[i][j] = rand() % 21 - 10 );

  createIndex( b, a, n, m );        /* tạo mảng ánh xạ chỉ số */
  printf( "Nhap buoc dich: " );
  scanf( "%d", &k );
  if ( k > n * m ) k = 0;
  rShiftkStep( a, b, n, m, k );     /* dùng mảng ánh xạ chỉ số */
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
    for ( j = 0; j < m; ++j )
      printf( "%5d", a[i][j] );
  return 0;
}
```

Bài 112: (trang 34)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

double fround( double x, long n ) {
  long round;
  x *= pow( 10, n );
  round = ( x > 0 ) ? (long)( x + 0.5 ) : (long)( x - 0.5 );
  return round / pow( 10, n );
}

void swaprow( double** a, int r1, int r2 ) {
  double* t = a[r1]; a[r1] = a[r2]; a[r2] = t;
}

int main() {
  double** a, t;
  int n, i, j, k;

  printf( "Nhap bac cua ma tran: ");
  scanf( "%d", &n );

  a = ( double ** )calloc( n, sizeof( double* ) );
  if ( !a )
    { printf( "Loi cap phat\n" ); return 1; }
  a[0] =  ( double * )calloc( n * n, sizeof( double ) );
  if ( !a[0] )
    { printf( "Loi cap phat\n" ); free( a ); return 1; }

  srand( time( NULL ) );
  for ( i = 0; i < n; ++i ) {
    a[i] = a[0] + i * n;
    for ( j = 0; j < n; ++j )
      printf( "%8.3lf", a[i][j] = rand()/(double)RAND_MAX );
    putchar( '\n' );
  }
  /* phép khử Gauss */
  for ( i = 0; i < n; ++i ) {
    if ( a[i][i] )
      for ( j = i + 1; j < n; ++j ) {
        t = a[j][i] / a[i][i];
        for ( k = 0; k < n; ++k )
          a[j][k] -= a[i][k] * t;
      }
    else {
      /* Kiểm tra xem trong trường hợp a[i][i] bằng 0,
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
          có tìm được a[j][i] nào khác 0 không */
      for ( j = i + 1; j < n; ++j )
        if ( a[j][i] ) break;
      if ( j < n ) swaprow( a, i, j );
      else break;
      i--;
    }
  }
  putchar( '\n' );

  if ( i >= n ) {
    for ( i = 0; i < n; ++i, putchar( '\n' ) )
      for ( j = 0; j < n; ++j )
        printf( "%8.3lf", fround( a[i][j], 3 ) );
  } else  printf( "Dinh thuc ma tran bang 0\n" );

  free( a[0] );
  free( a );
  return 0;
}
```

Hàm `rand()` sinh ra một số nguyên ngẫu nhiên trong đoạn `[0, RAND_MAX]`. Ta dùng `srand()` và `rand()` để sinh số thực ngẫu nhiên `x` như sau:

```
srand( time( NULL ) );
x = rand() / ( double )RAND_MAX;
```

Sau khi có ma trận các số thực ngẫu nhiên, ta tiến hành chính xác các bước của phép khử Gauss như đã mô tả trong phần hướng dẫn.
Hàm `fround()` dùng làm tròn các số thực trong ma trận kết quả, đã được trình bày trong bài tập 28 (trang 90).

Bài 113: (trang 35)

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

int main() {
  char s[255];
  int i, head;
  int maxhead, maxlen;

  printf( "Nhap chuoi nhi phan: " );
  fgets( s, 255, stdin );

  head = maxhead = maxlen = 0;
  do {
    int len = 0;
    for ( i = head; s[i] && s[i] == '0'; ++i )
      len++;
    if ( len > maxlen ) { maxlen = len; maxhead = head; }
    head = i + 1;
  } while ( s[i] );

  printf( "Chuoi 0 dai nhat co %d ky tu\n", maxlen );
  printf( "Bat dau tai s[%d] ", maxhead );
  putchar( '\n' );
  return 0;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
}
```

Hàm `gets( char* str )` đọc các ký tự từ `stdin` (ngõ nhập chuẩn) và đặt chúng vào mảng ký tự chỉ bởi con trỏ *`str`*, ký tự đọc vào cho đến khi xuất hiện một ký tự xuống dòng (newline) hoặc `EOF`. Ký tự newline không lưu trong mảng mà được thay thế bằng ký tự null kết thúc mảng. Tuy nhiên, hàm `gets()` không cung cấp kiểm tra giới hạn nhập (bounds checking) nên có thể gây tràn mảng ký tự nên được xem là "nguy hiểm" và khuyến cáo không nên dùng.

Để thay thế, đề nghị dùng hàm `fgets( char* str, int num, FILE* stream );` hàm này đọc tối đa *`num`* `- 1` ký tự từ *`stream`* (thường dùng là `stdin`) vào mảng ký tự chỉ bởi con trỏ *`str`*, ký tự đọc vào cho đến khi xuất hiện một ký tự xuống dòng (newline) hoặc `EOF`, ký tự newline *cũng được lưu trong mảng*. Sau đó một ký tự null kết thúc mảng được lưu ngay sau ký tự nhập cuối cùng.

Do `fgets()` không loại bỏ ký tự newline khi nhập, ta phải làm điều đó như sau:

```
char s[50];
size_t len;
printf( "Nhap mot chuoi: " );
fgets( s, 50, stdin );
len = strlen( s ) - 1;
if ( s[len] == '\n' ) s[len] = '\0';
```

hoặc gọn hơn, gọi hàm `strchr()`:

```
char s[50], *t;
printf( "Nhap mot chuoi: " );
fgets( s, 50, stdin );
if ( ( t = strrchr( s, '\n' ) ) != NULL ) *t = '\0';
```

Trở lại bài tập trên, thực chất ta cần tìm "run" dài nhất chứa toàn ký tự `'0'`, cách tìm tương tự như thực hiện trong mảng một chiều, xem bài tập 69 (trang 126), chỉ khác tính chất của "run" và cách kiểm tra đã đến cuối mảng ký tự.

C không hỗ trợ kiểu dữ liệu chuỗi (string). Trong C, chuỗi là mảng các ký tự và C có một dấu đặc biệt để chỉ rõ ký tự cuối trong một chuỗi. Ký tự này gọi là null và viết là `'\0'`. Bạn đừng nhầm lẫn ký tự null này với con trỏ `NULL` cũng như số `0` (kiểu `int`), ký tự null (kiểu `char`) luôn luôn bằng `0`.

Cũng tránh hiểu quá đơn giản: "chuỗi là mảng các ký tự", ví dụ một trường hợp đặc biệt xảy ra khi thiết lập chính xác số ký tự được dùng trong khi khởi tạo mà không tính đến ký tự null, như sau: `char characters[7] = "No null";`

Ở đây trình biên dịch cân nhắc việc thiếu ký tự kết thúc null. Trình biên dịch xác định đây là một *mảng ký tự*, không phải là một chuỗi.

Khi làm việc với chuỗi, việc kiểm tra kết thúc chuỗi không phải bằng cách so sánh với chiều dài chuỗi (`i < strlen( s )`) mà nên kiểm tra ký tự đang thao tác có khác null hay chưa, nghĩa là `s[i] != '\0'` hay gọn hơn (`s[i]`).

Bài 114: (trang 35)

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

size_t _strlen( const char *s ) {
  const char* p = s;
  while ( *p ) ++p;
  return p - s;
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
char* _strcpy( char *s, const char *t ) {
  char* p = s;
  while ( *p++ = *t++ ) {}
  return s;
}

char* _strcat( char *s, const char *t ) {
  char *p = s;
  while ( *p ) ++p;
  while ( *p++ = *t++ ) {}
  return s;
}

int main() {
  char* buf;
  size_t size;
  char s1[] = "the quick brown fox ";
  char s2[] = "jumps over the lazy dog";

  size = sizeof( s1 ) - 1 + sizeof( s2 ) - 1 + 1;
  if ( !( buf = ( char* )malloc( size ) ) )
    { printf( "Loi cap phat!" ); return 1; }

  printf( "Chuoi 1: [%s] (%lu)\n", s1, ( unsigned long )_strlen( s1 ) );
  printf( "Chuoi 2: [%s] (%lu)\n", s2, ( unsigned long )_strlen( s2 ) );
  printf( "strcpy( buf, s1 ) roi strcat( buf, s2 ):\n" );
  _strcat( _strcpy( buf, s1 ), s2 );
  printf( "[%s] (%lu)\n", buf, ( unsigned long )_strlen( buf ) );
  free( buf );
  return 0;
}
```

Một đặc điểm trong việc xử lý chuỗi là thực hiện các thao tác chủ yếu bằng con trỏ. Dùng con trỏ ký tự trong trường hợp này linh động và trực quan hơn nhiều so với dùng cách dùng chỉ số:

- Hàm `_strlen()`: mô phỏng `strlen()`, con trỏ `p` được khởi tạo chỉ đến đầu mảng `s`, nếu `p` vẫn chưa chỉ đến phần tử cuối mảng (`*p != '\0'`) thì ta còn tăng `p`. Nếu `p` đã chỉ đến phần tử cuối mảng, dùng phép trừ con trỏ để xác định số phần tử giữa `p` và đầu mảng.

Con trỏ trong C rất thuận lợi cho việc truy xuất các phần tử của mảng. Phép cộng con trỏ tỷ lệ theo kích thước phần tử của mảng do con trỏ chỉ đến, ví dụ `p++` có nghĩa là con trỏ `p` chỉ đến phần tử kế tiếp của mảng. Khi hai con trỏ cùng chỉ vào một mảng được trừ nhau, kết quả sẽ là số phần tử của mảng giữa chúng.

Để tránh một số nhầm lẫn, bạn ôn tập một chút về sử dụng con trỏ:

Ví dụ 1: `*p++` có nghĩa: lấy trị nơi con trỏ chỉ đến (`*p`), tăng *con trỏ* lên `1` (`p++`), con trỏ chỉ đến phần tử kế tiếp. Mặc dù toán tử `++` có độ ưu tiên cao hơn toán tử `*`, nhưng vì toán tử "tính sau" postfix `++` dùng thanh ghi lưu trị nên ta có cảm giác việc lấy trị nơi con trỏ chỉ đến được thực hiện trước.

Ví dụ 2: `(*p)++` có nghĩa: lấy trị nơi con trỏ chỉ đến (`*p`), rồi tăng *trị nơi con trỏ chỉ đến* lên `1` (`()++`), con trỏ không dịch chuyển.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Ví dụ 3: `*++p` có nghĩa: tăng *con trỏ* lên `1` (`++p`) do dùng toán tử "tăng trước" prefix, con trỏ chỉ đến phần tử kế tiếp, rồi lấy trị nơi con trỏ chỉ đến (`*p`).

- Hàm `_strcpy()`: mô phỏng `strcpy()`, con trỏ `p` được khởi tạo chỉ đến đầu mảng `s`. Trong vòng lặp, lần lượt các trị của phần tử nơi con trỏ `t` chỉ đến được sao chép sang phần tử nơi con trỏ `p` chỉ đến. Mỗi lần sao chép, hai con trỏ `p` và `t` đều tăng một đơn vị, nghĩa là sẽ chỉ đến phần tử kế tiếp. Trị vừa sao chép (`*p`) được kiểm tra để phát hiện ký tự null để chấm dứt vòng lặp. Như vậy, quá trình sao chép sẽ chấm dứt khi ký tự null cuối chuỗi `t` được sao chép sang `p` rồi được kiểm tra phát hiện thấy.

Ta nhận thấy con trỏ `t` được truyền bằng trị (by value) cho hàm `_strcpy()`. Nghĩa là con trỏ `t` được dùng bên trong hàm chỉ là bản sao, có thể thay đổi nó (`t++`) mà không ảnh hưởng đến chuỗi được truyền.

- Hàm `_strcat()`: mô phỏng `strcat( char* s1, const char* s2 )`.

Hàm `strcat()` sẽ nối một bản sao của chuỗi `s2` vào cuối chuỗi `s1`, ký tự đầu tiên của chuỗi `s2` sẽ chồng lên ký tự null cuối chuỗi `s1`. Chú ý là hàm `strcat()` không có kiểm tra biên (bounds checking), lập trình viên phải bảo đảm rằng chuỗi `s1` đủ lớn để chứa cả nội dung chuỗi gốc và chuỗi `s2` thêm vào.

Hàm `_strcat( s, t )` do chúng ta viết có hai thao tác: dùng con trỏ `p` đi về cuối chuỗi `s` (như hàm `_strlen()`), sau đó sao chép các phần tử từ `t` sang `p` (như hàm `_strcpy()`).

Với khai báo: `char s[] = "she see sea";`

Trình biên dịch sẽ tự tính chiều dài của chuỗi khai báo, kể cả ký tự `'\0'` (tức `12` ký tự). Có thể tính được chiều dài của chuỗi `s` bằng toán tử `sizeof`:

`( ( sizeof s ) / sizeof *s ) ) - 1`

toán tử `sizeof` tính luôn cả ký tự `'\0'` nên phải trừ kết quả đi `1` cho giống với kết quả của hàm `strlen()`. Vì `char` luôn có kích thước `1` byte nên có thể viết gọn hơn: `sizeof( s ) - 1`

Cũng do C chấp nhận kiểu `char` có kích thước `1` byte, nên khi cấp phát cho con trỏ ký tự với hàm `malloc()` ta không cần dùng toán tử `sizeof` nữa.

Bài 115: (trang 35)

```
#include <stdio.h>

int _strcmp( const char *s, const char *t ) {
  for ( ; *s == *t; ++s, ++t )
    if ( *s == '\0') return 0;
  return *s - *t;
}

char* _strchr( char *s, int c ) {
  for ( ; *s; s++ )
    if ( *s == c ) return s;
  return NULL;
}

char* _strrchr( char *s, int c ) {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  char* p = NULL;
  for ( ; *s; ++s )
    if ( *s == c ) p = s;
  return p;
}

int main() {
  char str[] = "jackdaws love my big sphinx of quartz";
  char *test[] = { "brown", "blue", "black" };
  int n = sizeof test / sizeof *test;
  int i, j;

  printf( "Chuoi goc s: [%s]\n", str );
  printf( "strchr( s, 'm' ) : [%s]\n", _strchr( str, 'm' ) );
  printf( "strrchr( s, 'o' ): [%s]\n", _strrchr( str, 'o' ) );

  printf( "Sap xep cac chuoi dung strcmp():\n" );
  for ( i = 0; i < n; ++i )
    for ( j = i + 1; j < n; ++j )
      if ( _strcmp( test[i], test[j] ) > 0 ) {
        char *t = test[i];
        test[i] = test[j];
        test[j] = t;
      }
  for ( i = 0; i < n; ++i )
    printf( "%s  ", test[i] );
  putchar( '\n' );
  return 0;
}
```

Ta thực hiện các hàm mô phỏng như sau:

- Hàm `_strcmp()`:

Mô phỏng `strcmp( const char* s1, const char* s2 )`. Hàm `strcmp()` lần lượt so sánh các ký tự trong hai chuỗi được truyền tới hàm tại những vị trí tương ứng theo thứ tự từ đầu chuỗi đến cuối chuỗi. Nếu trong quá trình so sánh, phát hiện ký tự không so trùng tại vị trí `k` tương ứng trên hai chuỗi, hàm trả về:

trị âm nếu `s1[k] < s2[k]`.

trị dương nếu `s1[k] > s2[k]`.

trị `0` nếu hai chuỗi giống nhau.

C không quy định cụ thể trị trả về, tùy cách cài đặt hàm ta nhận được trị trả về là `(-1, 0, 1)` hoặc `(-d, 0, d)` (`d` là khoảng cách trong bảng mã `ASCII` giữa hai ký tự được phát hiện khác nhau).

Ta cài đặt như sau: vòng lặp `for` sẽ lặp khi hai ký tự tại vị trí tương ứng trên hai chuỗi còn giống nhau. Hai ký tự này do hai con trỏ `s` và `t` chỉ đến, `s` và `t` tăng dần sau mỗi vòng lặp. Chú ý `s` và `t` là biến cục bộ của hàm, trị của chúng được truyền bằng trị đến hàm, nên sự di chuyển của chúng không ảnh hưởng đến hai chuỗi. Đến khi phát hiện `*s = '\0'`, và dĩ nhiên lúc đó `*t` cũng bằng `'\0'`, vì nếu không vòng lặp đã ngắt không vào đến thân vòng lặp, trả về `0` để xác định hai chuỗi giống nhau. Nếu phát hiện hai ký tự khác nhau (`*s != *t`) hoặc một trong hai chuỗi chấm dứt, ví dụ chuỗi `t` (`*s != '\0'`), vòng lặp sẽ ngắt.

Với cách cài đặt trị trả về `(-1, 0, 1)` ta trả về: `return *s > *t ? 1 : -1;`

Với cách cài đặt trị trả về `(-d, 0, d)` ta trả về: `return *s - *t;`

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Cần nhớ là `strcmp()` dùng so sánh các chuỗi, không phải các ký tự. Ký tự, kiểu `char`, được C xem như kiểu `int`, có trị đồng nhất với vị trí của chúng trong bảng mã `ASCII`, nên có thể so sánh chúng bằng các toán tử so sánh thông thường. Trong ví dụ minh họa cách dùng hàm `_strcmp()`, ta sắp xếp mảng các chuỗi theo giải thuật Selection Sort với điều kiện sắp xếp là so sánh giữa hai chuỗi bằng `_strcmp()`.

- Hàm `_strchr()`:

Hàm `strchr( const char *s, int c );` trả về một con trỏ chỉ đến ký tự `c` (được chuyển kiểu ngầm thành `char`) đầu tiên xuất hiện trong chuỗi `s` (ký tự null kết thúc chuỗi vì thuộc chuỗi nên cũng có thể được tìm). Nếu không phát hiện ra ký tự `c`, hàm trả về một con trỏ `NULL`.

Ta cài đặt như sau: vòng lặp `for` cứ tăng `s` cho đến khi phát hiện ký tự `c` (trả về địa chỉ `s` của ký tự `c`) hoặc đến cuối chuỗi (`*s = '\0'`). Nếu đã vượt qua vòng lặp kiểm tra (chưa trả về vì chưa phát hiện ký tự `c`), ta trả về con trỏ `NULL`.

- Hàm `_strrchr()`:

Mô phỏng hàm `strrchr( const char *s, int c )`. Giống như hàm `strchr()` nhưng trả về một con trỏ chỉ đến ký tự `c` xuất hiện cuối cùng trong chuỗi `s`. Nếu không phát hiện ra ký tự `c`, hàm trả về một con trỏ `NULL`.

Có vẻ như nếu tiến hành tìm kiếm từ cuối chuỗi thì sẽ nhanh hơn. Tuy nhiên, vẫn phải đi đến cuối chuỗi bằng vòng lặp hoặc gọi hàm (`strlen()`, về bản chất cũng tương tự). Ta cài đặt như sau: chuẩn bị một con trỏ `p` chứa kết quả, khởi tạo bằng `NULL`. Tiến hành tìm kiếm `c` từ đầu chuỗi giống như hàm `_strchr()`, nhưng khi tìm được thì không trả địa chỉ tìm được về mà lưu kết quả vào `p`, chồng lên kết quả cũ. Cứ như thế cho đến khi kết thúc vòng lặp, `p` sẽ chứa địa chỉ cuối cùng tìm thấy `c` hoặc vẫn là `NULL` nếu không tìm thấy `c`; trả `p` về.

Bài 116: (trang 35)

```
#include <stdio.h>

size_t _strlen( const char *s ) {
  const char* p = s;
  while ( *p++ ) {}
  return p - s - 1;
}

const char* _strchr( const char *s, int c ) {
  int i;
  for ( i = 0; s[i]; ++i )
    if ( s[i] == c ) return s + i;
  return NULL;
}

size_t _strspn( const char* s1, const char* s2 ) {
  size_t i;
  for ( i = 0; s1[i] && _strchr( s2, s1[i] ); ++i ) { }
  return i;
}

int _strncmp( const char *s1, const char *s2, size_t len ) {
  size_t i;
  if ( len > _strlen( s1 ) ) len = _strlen( s1 );
  if ( len > _strlen( s2 ) ) len = _strlen( s2 );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  for ( i = 0; i < len; ++i )
    if ( s1[i] != s2[i] )
      return s1[i] > s2[i] ? 1 : -1;
  return 0;
}

const char* _strstr( const char *s1, const char *s2 ) {
  size_t len = _strlen( s2 );
  for ( ; *s1; ++s1 )
    if ( _strncmp( s1, s2, len ) == 0 ) return s1;
  return NULL;
}

int main() {
  char s1[] = "hom qua qua noi qua ma qua khong qua";
  const char* p = s1;
  const char* oldp = s1;
  char s2[] = "cabbage";

  printf( "Chuoi kiem tra : %s\n", s1 );
  printf( "Vi tri tu 'qua': " );
  while ( ( p = _strstr( p, "qua" ) ) != NULL ) {
    printf( "%*c", p - oldp + 1, 'x' );
    oldp = ++p;
  }
  printf( "\nKy tu dau tien cua s = '%s' khong co trong '%s' la s[%lu]\n",
          s2, "abc", ( unsigned long )_strspn( s2, "abc" ) );
  return 0;
}
```

Các hàm mô phỏng trong bài tập này chủ ý dùng chỉ số nên có vẻ dễ hiểu với bạn, nhưng bạn cần nhớ rằng thao tác trên chuỗi chủ yếu được thực hiện bằng con trỏ. Các bài tập mô phỏng lại các hàm trong `string.h` nhằm mục đích nắm vững các thao tác chi tiết trên chuỗi.

- Hàm `_strspn()`: mô phỏng hàm `strspn( s1, s2 )`, hàm này được mô tả khá khó hiểu trong tài liệu, thực tế được dùng để tìm chỉ số của phần tử đầu tiên trong chuỗi `s1` sao cho ký tự đó không có mặt trong chuỗi `s2`.

Ta cài đặt như sau: vòng lặp `for` duyệt các phần tử của chuỗi `s1`, chỉ bị ngắt bởi hai trường hợp: đã đến cuối chuỗi (`s1[i] = '\0'`) hoặc phát hiện ra ký tự của `s[1]` không có mặt trong `s2` (`strchar( s2, s1[i] ) == NULL`). Trị trả về là chỉ số `i` của phần tử làm vòng lặp bị ngắt.

- Hàm `_strncmp()`: mô phỏng hàm `strncmp( s1, s2, n )`, hàm này giống hàm `strcmp()` nhưng so sánh sẽ dừng sau khi kiểm tra `n` ký tự đầu tiên hoặc khi phát hiện ra ký tự null đầu tiên. Nghĩa là nếu có chuỗi so sánh ngắn hơn `n`, hàm chỉ tiến hành so sánh `len` ký tự với `len` là kích thước chuỗi ngắn hơn.

Ta cài đặt như sau: trước hết ta xác định lại `n`, sẽ bằng chiều dài chuỗi ngắn hơn, nếu có chuỗi ngắn hơn `n`. Sau đó ta dùng vòng lặp `for` duyệt `n` phần tử của chuỗi `s1`, dừng khi phát hiện ký tự đang xét khác ký tự ở vị trí tương ứng trong chuỗi `s2`, trả về trị âm hoặc dương tùy kết quả so sánh. Nếu vượt qua vòng lặp kiểm tra, so sánh đã thành công và hàm trả về trị `0`.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

- Hàm `_strstr()`: mô phỏng hàm `strstr( s1, s2 )`; nếu chuỗi con `s2` có mặt trong `s1`, hàm trả về vị trí xuất hiện đầu tiên của chuỗi `s2` trong `s1`, ngược lại, hàm trả về con trỏ `NULL`.

Ta cài đặt như sau: trong vòng lặp `for` với `s1` tăng liên tục, ta dùng hàm `strncmp()` để so sánh `s1` với `s2` trong `strlen(s2)` ký tự. Nếu thành công, trả về `s1`; ngược lại, tăng `s1` và tiếp tục so sánh lại. Nếu vượt qua vòng lặp mà vẫn chưa phát hiện được chuỗi `s2` trong `s1`, trả về con trỏ `NULL`.

Cần nhắc lại rằng: một phần của chuỗi, bắt đầu từ một ký tự nào đó trước ký tự null kết thúc chuỗi, vẫn được xem như một chuỗi:

```
char s[] = "she see sea";
char* p = s;
puts( p + 4 );       /* chuỗi con: "see sea" */
puts( p += 8 );      /* chuỗi con: "sea" */
```

Đa số các hàm xử lý chuỗi của C nhận tham số là một con trỏ chỉ đến chuỗi cần xử lý, trả về một con trỏ đến chuỗi con hoặc `NULL` nếu không thành công. Vì vậy, C thường dùng một "vòng lặp xử lý chuỗi điển hình", được minh họa bên dưới.

```
/* vòng lặp xử lý chuỗi điển hình của C: in vị trí xuất hiện chuỗi "qua" trong s */
char* p = s;
                                  1) p đầu tiên chính là chuỗi cần xử lý
while ( ( p = _strstr( p, "qua" ) ) != NULL )
{ 2) dùng kết
  quả tìm được
  printf( "s[%d] ", p - s );      3) cập nhật lại p
  p++;                               để xử lý tiếp
}                                                    4) dừng vòng lặp
                                                     nếu không tìm được
```

Để sử dụng hiệu quả các hàm trong `string.h`, bạn cần đặc biệt hiểu rõ cách xây dựng vòng lặp này.

Kinh nghiệm: nếu cần xây dựng một hàm xử lý chuỗi nào đó bổ sung cho các hàm của `string.h`, ta xây dựng giống như các hàm xử lý chuỗi của C: nhận `char*`, trả về `char*` hoặc `NULL` nếu không thành công.

Trong ví dụ minh họa dùng chuỗi định dạng `"%*c"` để "dành chỗ", đẩy ký tự `x` chỉ từ `"qua"` đến vị trí thích hợp, xem bài tập 28 (trang 90).

Hàm `_strstr()` do chúng ta viết lẫn hàm `strstr()` của `string.h` đều tìm chuỗi với chế độ so sánh nghiêm ngặt, phân biệt chữ hoa và chữ thường (case sensitive). Để có thể tìm chuỗi với chế độ insensitive, dùng hàm sau:

```
#define islower( c ) ( (c) >= 'a' && (c) <= 'z' )
#define toupper( c ) ( islower((c))? (c) + ('A'-'a') : (c) )
const char* _stristr( const char *s1, const char *s2 ) {
  for ( ; *s1; ++s1 ) {
    if ( toupper( *s1 ) == toupper( *s2 ) ) {
      const char *h, *n;
      for ( h = s1, n = s2; *h && *n; ++h, ++n )
        if ( toupper( *h ) != toupper( *n ) ) break;
      if ( !*n ) return s1;
    }
  }
  return NULL;
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Một giải pháp khác, vẫn dùng `strstr()`, là chuyển thành chữ hoa cả hai chuỗi tham số. Ta dùng hàm `strupr()`, tuy nhiên cần nhớ hàm `strupr()` sẽ chuyển chuỗi nó nhận thành chữ hoa luôn nên cần dùng `strdup()` để sao chép chuỗi trước khi gọi.

Bài tập: Viết hàm `findByKeyword()` tìm trong mảng các structure `Student` những `Student` có trường `name` chứa `keyword` nhập từ bàn phím.

```
#define MAX 20
typedef struct {
  int code;
  char name[20];
  float mark;
} Student;

void findByKeyword( Student students[], int size ) {
  char keyword[20];
  char* t;
  int i, rsize = 0;
  Student results[MAX];
  printf( "Keyword? " );
  fgets( keyword, 20, stdin );
  if ( ( t = strrchr( strupr( keyword ), '\n') ) != NULL ) *t = '\0';
  for ( i = 0; i < size; ++i ) {
    t = strupr( strdup( students[i].name ) );
    if ( strstr( t, keyword ) != NULL )
      results[rsize++] = students[i];
  }
  if ( rsize == 0 ) printf( "Student not found!\n" );
  else {
    printf( "%-7s %-20s %s\n", "Code", "Name", "Mark" );
    printf( "----------------------------------\n" );
    for ( i = 0; i < rsize; ++i )
      printf( "%-7d %-20s %g\n",
              results[i].code, results[i].name, results[i].mark );
  }
}
```

Bài 117: (trang 35)

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

char* left( const char* str, size_t n ) {
  char* buf;
  size_t len = strlen( str );
  if ( n > len ) n = len;
  buf = ( char* ) malloc( n + 1 );
  buf[n] = '\0';
  return strncpy( buf, str, n );
}

char* right( const char* str, size_t n ) {
  char *buf;
  size_t len = strlen( str );
  if ( n > len ) n = len;
  buf  = strdup( &str[len - n] );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  return buf;
}

int main() {
  char str[] = "Kernighan and Ritchie";

  printf( "Chuoi goc: [%s]\n", str );
  printf( "left( str, 9 ) : [%s]\n", left( str, 9 ) );
  printf( "right( str, 7 ): [%s]\n", right( str, 7 ) );
  return 0;
}
```

Cả hai hàm đều: xử lý lại chiều dài `n` được truyền cho hợp lý, cấp phát chuỗi kết quả mới với chiều dài `n + 1`, cuối cùng sao chép chuỗi con tách được sang chuỗi kết quả. Tuy nhiên:

- Hàm `left()`: `strncpy()` bỏ qua ký tự null kết thúc, nên chúng ta cần thêm ký tự null vào cuối để tạo ký tự kết thúc cho chuỗi kết quả.
- Hàm `right()`: chú ý `str` là một chuỗi thì `&str[len - n]` cũng là một chuỗi. C đều hiểu chúng như địa chỉ: `str` và `str + len - n`. Chuỗi `&str[len - n]` được sao chép đã có ký tự kết thúc null nên không cần quan tâm nữa.

Bài 118: (trang 35)

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

char* allTrim( char* s ) {
  char* p = s;
  /* tìm chuỗi có hai space liên tiếp */
  while ( ( p = strstr( p, "  " ) ) != NULL )
    strcpy( p, p + 1 );
  if ( *s == ' ' ) strcpy( s, s + 1 );
  p = s + strlen( s ) - 1;
  if ( *p == ' ' ) *p = '\0';
  return s;
}

char* stdWord( char* s ) {
  int i;
  if ( !s ) return NULL;
  s[0] = toupper( s[0] );
  for ( i = 1; s[i]; ++i )
    s[i] = tolower( s[i] );
  return s;
}

int main() {
  char str[] = "  'bJARne?  sTROUstRUP'   ";
  char d[] = " \t\"'?!.,";
  char *p = str;

  printf( "Chuoi goc    : [%s]\n", str );
  printf( "Loai space du: [%s]\n", allTrim( str ) );
  printf( "Cac tu da chuan hoa:\n" );
  while ( ( p = stdWord( strtok( p, d ) ) ) != NULL ) {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    printf( "   %s\n",  p  );
    p = NULL;
  }
  return 0;
}
```

Chuẩn hóa chuỗi là tiến hành các công việc:
- Xóa tất cả các ký tự trắng đầu chuỗi, (thường gọi là `ltrim`).
- Xóa tất cả các ký tự trắng cuối chuỗi, (thường gọi là `rtrim`).
- Với các ký tự trắng ở giữa chuỗi, xóa và chỉ chừa lại đúng một ký tự trắng (thường gọi là `mtrim`).

Ta sẽ thực hiện cả ba thao tác xử lý trên trong hàm `allTrim()`.
- Các ký tự đầu mỗi từ được viết hoa, các ký tự còn lại được viết thường, (thường gọi là capitalize hay title case). Thao tác xử lý mỗi từ của chuỗi được thực hiện trong hàm `stdWord()`.

Hàm `allTrim( s )` được cài đặt như sau:
- Tìm tất cả các chuỗi con có 2 space liên tiếp trong `s` và xóa đi ký tự space đứng trước trong cặp space đó bằng hàm `strcpy()`:

```
strcpy( p, p + 1 ); /* xóa 1 ký tự nơi p chỉ đến */
```

- Có thể còn một ký tự space đầu chuỗi, kiểm tra và lại xóa bằng hàm `strcpy()`.
- Có thể còn một ký tự space cuối chuỗi, kiểm tra và đơn giản đặt lại ký tự này bằng ký tự null kết thúc chuỗi.

Hàm `stdWord()` cài đặt như sau:
- Chuyển các ký tự đầu từ thành ký tự hoa bằng hàm `toupper()` của `ctype.h`.
- Dùng vòng lặp chuyển các ký tự còn lại của từ thành ký tự thường bằng hàm `tolower()` của `ctype.h`.

Để xử lý tất cả các từ của chuỗi ta phải tách các từ này ra bằng hàm `strtok()`.
Hàm `strtok( char *s1, const char *s2 );` trả về một con trỏ chỉ đến token (chuỗi ký tự theo một quy tắc nào đó) kế tiếp trong chuỗi chỉ bởi con trỏ `s1`, các ký tự mô tả trong chuỗi `s2` được xem như là các ký tự ngăn cách (delimiter) giữa các token. Khi không có token nào được phát hiện, hàm trả về con trỏ `NULL`.
Trong quá trình tách token (tokenized), hàm `strtok()` lưu một con trỏ nội chỉ đến chuỗi cần tách token. Khi `strtok()` được gọi đầu tiên với tham số thứ nhất là một chuỗi, con trỏ nội được thiết lập chỉ đến chuỗi này. Trong chuỗi các lần gọi sau, tham số thứ nhất là `NULL`, con trỏ nội `strtok()` sẽ tiếp tục tách các token trong chuỗi, từ sau token tách lần đầu.
Như vậy để tách các từ của chuỗi ta chia làm hai giai đoạn gọi hàm `strtok()`:

```
printf( "   %s\n", stdWord( strtok( str, d ) ) );
while ( ( p = stdWord( strtok( NULL, d ) ) ) != NULL )
  printf( "   %s\n",  p  );
```

hoặc dùng "vòng lặp xử lý chuỗi điển hình" như trong bài 116 (trang 186):

```
char* p = s;
while ( ( p = stdWord( strtok( p, d ) ) ) != NULL ) {
  printf( "   %s\n",  p  );
  p = NULL;
}
```

Bài 119: (trang 36)

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

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


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

int main() {
  char s[255];
  char* p = s;
  int freq[7] = { 0 };
  int i, count;

  printf( "Nhap chuoi: " );
  fgets( s, 255, stdin );
  count = 0;
  while ( ( p = strtok( p, ".,  " ) ) != NULL ) {
    if ( isalpha( p[0] ) ) {
      count++;
      freq[strlen( p ) - 1]++;
    }
    p = NULL;
  }
  printf( "Co %d tu\n", count );
  printf( "Tan so xuat hien cac tu:\n" );
  for ( i = 0; i < 7; ++i )
    printf( "%d[%d]  ", i + 1, freq[i] );
  putchar( '\n' );
  return 0;
}
```

Bài tập này đơn giản dùng "vòng lặp xử lý chuỗi điển hình" kết hợp với hàm `strtok()` (xem bài 118, trang 190) để tách các từ trong chuỗi nhập.
Các từ trong chuỗi nhập, nếu không phải số, sẽ được đếm từ và đếm tần số. Tần số xuất hiện các từ có cùng số ký tự lưu trong mảng `freq[7]` (từ dài nhất của tiếng Việt, từ "nghiêng", chỉ có 7 ký tự), số các từ có `k` ký tự sẽ được lưu tại phần tử có chỉ số `k - 1`.

Bài 120: (trang 36)

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

char* insertString( char* s, char* s1, size_t k ) {
  char* s2 = strdup( s );
  if ( k > strlen( s ) ) k = strlen( s );
  strcpy( &s2[k], s1 );
  strcpy( &s2[k + strlen( s1 )], &s[k] );
  return s2;
}

int main() {
  char s[80], s1[80], *t;
  unsigned long k;

  printf( "Chuoi goc  : " );
  fgets( s, 80, stdin );
  if ( ( t = strrchr( s, '\n' ) ) != NULL ) *t = '\0';
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  printf( "Chuoi chen : " );
  fgets( s1, 80, stdin );
  if ( ( t = strrchr( s1, '\n' ) ) != NULL ) *t = '\0';

  printf( "Vi tri chen: " );
  scanf( "%lu", &k );
  printf( "Chuoi ket qua: %s\n", insertString( s, s1, (size_t) k ) );
  return 0;
}
```

Hình dưới mô tả cách thực hiện hàm `insertString()`:

- Trước hết chuỗi `s` được sao chép vào `s2` bằng `strdup()`.
- Tại vị trí `s2[k]` của chuỗi sao chép vào ở trên, sao chép chuỗi `s1` vào.
- Tại vị trí `s2[k + strlen( s1 )]`, tức tại ký tự null của chuỗi `s1` vừa sao chép, ta sao chép đoạn sau của chuỗi `s` (tính từ `k`, tức chuỗi `&s[k]`) vào tiếp. Ký tự null của chuỗi này cũng là ký tự null kết thúc chuỗi kết quả.

Bài 121: (trang 36)

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

int removeString( char *s, int start, int count ) {
  size_t len = strlen( s );
  if ( start > len || start < 0 || len < start + count )
    return 1;
  strcpy( &s[start], &s[start + count] );
  return 0;
}

int main() {
  char s[80], *t;
  int start, count;

  printf( "Nhap chuoi: " );
  fgets( s, 80, stdin );
  if ( ( t = strrchr( s, '\n' ) ) != NULL ) *t = '\0';
  printf( "Nhap vi tri dau: " );
  scanf( "%d", &start );
  printf( "Nhap so ky tu loai bo: " );
  scanf( "%d", &count );
  if ( removeString( s, start, count ) ) puts( "Error!" );
  else printf( "Chuoi ket qua: %s\n", s );
  return 0;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
}
```

Khi xóa một ký tự hoặc một đoạn trong chuỗi ta không dùng cách dịch chuyển phần tử như mảng mà dùng hàm `strcpy()` của `<string.h>`, về thực chất hàm này cũng dịch chuyển vùng nhớ. Khi sao chép chuỗi chồng lên đoạn chuỗi cần xóa, ký tự null kết thúc của chuỗi sao chép cũng là ký tự null kết thúc của chuỗi kết quả.

Bài 122: (trang 36)

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

long bin2dec( char* s ) {
  long d = 0;
  while ( *s ) d = ( d << 1 ) | ( *s++ - '0' );
  return d;
}

int main() {
  char s[255], *t;

  printf( "Nhap chuoi nhi phan: " );
  fgets( s, 255, stdin );
  if ( ( t = strrchr( s, '\n' ) ) != NULL ) *t = '\0';

  printf( "Tri thap phan: %ld\n", bin2dec( s ) );
  return 0;
}
```

Cho chuỗi nhị phân `s`, ta có trị thập phân tương ứng là:

$d = s[0].2^{n-1} + s[1].2^{n-2} + ... + s[n-2].2 + s[n-1]$

đặt: $k = n - 1$

$d = P(2) = s[0].2^{k} + s[1].2^{k-1} + ... + s[k-1].2 + s[k]$

Tính trị của đa thức này bằng phương pháp Horner, chỉ phải thực hiện `k = n-1` phép nhân.

$d = P(2) = (...((s[0].2 + s[1]).2 + s[2]).2 + ... + s[k-1]).2 + s[k]$

Chuyển thành vòng lặp, chú ý chuyển ký tự `s[i]` thành số: `s[i] - '0'`:

```
d = 0;
for ( i = 0; s[i]; ++i )
  d = d * 2 + ( s[i] - '0' );
```

hoặc:

```
while ( *s )
  d = d * 2 + ( *s++ - '0' );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

biểu diễn bằng "ngôn ngữ" của các toán tử bitwise:

```
while ( *s )
  d = ( d << 1 ) | ( *s++ - '0' );
```

Lập trình viên C có thói quen thao tác với các toán tử cấp thấp. Vào thời điểm C ra đời, Brian và Dennis làm việc với các toán tử bitwise nhiều hơn các toán tử luận lý, cho nên họ dùng 1 ký tự cho toán tử thao tác bit (ví dụ &) và 2 ký tự cho toán tử luận lý (ví dụ &&).

Bài tập: Thực hiện tác vụ ngược lại, chuyển số thập phân thành nhị phân.
Có sử dụng stack để dễ dàng thực hiện bằng đệ quy đầu.

```
void dec2bin( int n ) {
  if ( n / 2 == 0 ) printf( "%d", n % 2 );
  else {
    dec2bin( n / 2 );
    printf( "%d", n % 2 );
  }
}
```

hoặc không đệ quy, dùng mảng:

```
void dec2bin( int n ) {
  int bin[255], k = -1;
  do {
    bin[++k] = n % 2;
  } while ( ( n /= 2 ) > 0 );
  for( ; k >= 0; --k )
    printf( "%d", bin[k] );
}
```

Bài 123: (trang 36)

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

char* addBinStr( char* a, char* b ) {
  size_t alen, blen;
  int clen, t, carry;
  char *c;
  alen = strlen( a );
  blen = strlen( b );
  clen = ( alen > blen ) ? alen + 1 : blen + 1;
  c = ( char* )calloc( clen + 1, sizeof( char ) );
  if ( !c ) return NULL;
  carry = 0;
  /* a và b còn phần tử */
  while ( alen && blen ) {
    t = a[alen-1] - 48 + b[blen -1] - 48 + carry;
    carry = t / 2;
    c[clen-1] = (t % 2) + 48;
    alen--; blen--; clen--;
  }
  /* a còn phần tử */
  while ( alen ) {
    t = a[alen-1] - 48 + carry;
    carry = t / 2;
    c[clen-1] = (t % 2) + 48;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    alen--; clen--;
  }
  /* b còn phần tử */
  while ( blen ) {
    t = b[blen-1] - 48 + carry;
    carry = t / 2;
    c[clen-1] = (t % 2) + 48;
    blen--; clen--;
  }
  /* a và b đều hết phần tử, xét carry cuối */
  if ( carry ) c[clen-1] = 1 + 48;
  else c++;
  return c;
}

int main() {
  char a[] = "1101010000110001";     /* 54321 */
  char b[] =   "11000000111001";     /* 12345 */
  char* c;
  /* a + b = 10000010001101010          66666 */
  printf( "%20s\n", a );
  printf( "%20s\n", b );
  printf( "%20s\n", "-----------------" );
  printf( "%20s\n", c = addBinStr( a, b ) );
  free( c );
  return 0;
}
```

Hình bên dưới trình bày tóm tắt các trường hợp cần tính toán. Để dễ hiểu, trong hình các phần tử chứa số (`0` hoặc `1`). Hai chuỗi `a` và `b` chứa `2` số nhị phân `a`, `b`; chuỗi `c` chứa số nhị phân là tổng `a + b`.

Nếu `ch` là ký tự số, số tương ứng sẽ là `ch - 48` (hoặc `ch - '0'`) nếu `d` là số (1 chữ số), ký tự tương ứng là `d + 48` (hoặc `d + '0'`). Cần ghi nhớ điều này khi lấy một ký tự ra tính toán (chuyển ký tự thành số) hoặc khi ghi kết quả tính được vào chuỗi kết quả (chuyển số thành ký tự).

Quá trình tính toán tiến hành từ phải sang trái trong vòng lặp, có 3 trường hợp phải giải quyết trong hàm `addBinStr()`:

- Cả hai chuỗi `a` và `b` còn phần tử (hình trái):

```
c[i] = ( a[i] + b[i] + carry(cũ) ) % 2
carry(mới) = ( a[i] + b[i] + carry(cũ) ) / 2
```

- Chỉ một chuỗi (`a` hoặc `b`) còn phần tử (hình giữa): giả sử là `a`, tương tự với `b`

```
c[i] = ( a[i] + carry(cũ) ) % 2
carry(mới) = ( a[i] + carry(cũ) ) / 2
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

- Cả hai chuỗi a và b đều hết phần tử (hình phải): nếu còn carry thì ghi carry này vào c.

c[i] = carry(còn lại, bằng 1)

Khi cấp phát cho chuỗi c, ta dùng hàm calloc(). Mục tiêu là khởi tạo cho các phần tử của chuỗi c, đặc biệt là ký tự kết thúc chuỗi. Nếu cấp phát bằng hàm malloc(), bạn nhớ gán phần tử cuối trị c ký tự '\0' như sau:

```
c = ( char* )malloc( clen + 1 );
if ( !c ) return NULL;
c[clen] = '\0';
```

Bài 124: (trang 37)

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

void itos( char *s, int n ) {
  char *p;
  for ( p = s; n > 0; ++p, n /= 10 )
    *p = ( n % 10 ) + '0';
  if ( p == s ) {
    *p++ = '0';
    *p = '\0';
  } else {
    *p = '\0';
    for ( --p; p > s; ++s, --p )
      { char t = *p; *p = *s; *s = t; }
  }
}

int stoi( const char *s ) {
  int d = 0;
  while ( *s && !isdigit( *s ) ) s++;
  while ( isdigit( *s ) )
    d = d * 10 + *s++ - '0';
  return d;
}

int main() {
  int n;
  char s[100];

  printf( "Nhap so nguyen duong n: " );
  scanf( "%d", &n );

  itos( s, n );
  printf( "itos() -> %s\n", s );
  printf( "stoi() -> %d\n", stoi( s ) );
  return 0;
}
```

Hàm itos(): chúng ta tách các chữ số thập phân từ phải sang trái, chuyển thành ký tự rồi lưu vào chuỗi từ trái sang phải, ngược với thứ tự bình thường. Do không dùng mảng phụ, cấu trúc dữ liệu thích hợp (stack) hoặc đệ quy, nên chúng ta sẽ chọn cách đảo ngược chuỗi kết quả để có chuỗi với thứ tự bình thường.

Hàm itos() có hai giai đoạn:

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

- Vòng lặp với p, tách chữ số thập phân (n % 10) từ trái sang phải, chuyển thành ký tự (+ '0' hoặc + 48) rồi chuyển vào chuỗi tại vị trí do p chỉ đến. Kết thúc giai đoạn này cần chú ý nhập thêm ký tự null để kết thúc chuỗi.
- Đảo chuỗi: lúc này s chỉ phần tử đầu chuỗi và p chỉ phần tử cuối chuỗi (không phải phần tử null). Trong lúc hai con trỏ này tiến lại gần nhau (++s và --p), ta hoán chuyển nội dung của các phần tử do chúng chỉ đến với nhau để đảo ngược chuỗi.

Hàm stoi(): thực hiện tương tự bài 122 (trang 194), dùng phương pháp Horner để định trị đa thức:

d = P(10) = (...((s[0].10 + s[1]).10 + s[2]).10 + ... + s[k – 1]).10 + s[k]

chú ý chuyển ký tự của chuỗi thành số tương ứng (*s - '0'):

```
while ( isdigit( *s ) )
  d = d * 10 + ( *s++ - '0' );
```

Bài 125: (trang 37)

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

char* reverse( char* s ) {
  int i;
  size_t len = strlen( s );
  for ( i = 0; i < len / 2; ++i ) {
    char t = s[i];
    s[i] = s[len - i - 1];
    s[len - i - 1] = t;
  }
  return s;
}

void multiply( char* a, char* b ) {
  char c[20], d[20];
  size_t alen, blen;
  int i, j, carry, k, l, t;
  blen = strlen( b ) - 1;
  alen = strlen( a ) - 1;

  printf( "%20s\n", a );
  printf( "%3c\n", '*' );
  printf( "%20s\n", b );
  printf( "%s\n", "--------------------" );

  /* chuẩn bị chuỗi d (các ký tự đều là '0') để lưu tổng kết quả */
  for ( i = 0; i < 20; ++i ) d[i] = '0';

  /* l dùng trong printf để cân chỉnh kết quả in ra, dịch dần sang trái 1 đơn vị
     vòng lặp i tính một dòng nhân, cộng vào tổng kết quả, in dòng nhân ra */
  for ( l = 20, i = 0; b[i]; ++i, --l ) {  /* với mỗi số bị nhân */
    /* tạo chuỗi c, lưu kết quả (ngược) của một dòng nhân */
    for ( carry = j = 0; a[j]; ++j ) {    /* nhân cho số nhân */
      t = ( b[blen - i] - 48 ) * ( a[alen - j] - 48 ) + carry;
      carry = t / 10;
      c[j] = t % 10 + 48;
    }
    /* kiểm tra carry cuối để đóng chuỗi c, kết thúc một dòng nhân */
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    if ( carry ) { c[j] = carry + 48; c[++j] = '\0'; }
    else c[j] ='\0';

    /* cộng kết quả một dòng nhân từ c vào d, bắt đầu tại vị trí i
       tính từ trái sang (vì lưu ngược) đối với dòng nhân thứ i */
    for ( carry = k = 0; c[k]; ++k ) {
      t = ( d[k + i] - 48 ) + ( c[k] - 48 ) + carry;
      carry = t / 10;
      d[k + i] = ( t % 10 ) + 48;
    }
    /* k lưu vị trí cuối chuỗi d, dùng đóng d sau khi cộng tất cả các dòng nhân */
    if ( carry ) { d[k + i] = carry + 48; k += i + 1; }
    else k += i;
    /* in một dòng nhân, tức in c đảo ngược, cân chỉnh bằng l */
    printf( "%*s\n", l, reverse( c ) );
  }

  printf( "%s\n", "--------------------" );
  d[k] = '\0';          /* đóng chuỗi d tại k */
  /* in tổng kết quả, tức in d đảo ngược */
  printf( "%20s\n", reverse( d ) );
}

int main() {
  char a[20], b[20], *t;

  printf( "So bi nhan:  " );
  fgets( a, 20, stdin );
  if ( ( t = strrchr( a, '\n' ) ) != NULL ) *t = '\0';

  printf( "So nhan   :  " );
  fgets( b, 20, stdin );
  if ( ( t = strrchr( b, '\n' ) ) != NULL ) *t = '\0';

  multiply( a, b );
  return 0;
}
```

Trước khi thực hiện bài này, hãy đọc và hiểu thật rõ bài 123 (trang 195). Bài tập trên không khó, nhưng do những đặc điểm riêng khi thao tác với chuỗi, bạn cần thực hiện chính xác và cẩn thận. Xem chú thích chi tiết trong bài giải, chú ý các vấn đề sau:

- Phép nhân trong bài tập được thực hiện như sau: từng số của số bị nhân (chuỗi `a`, từ phải sang trái) sẽ nhân với chuỗi số nhân (chuỗi `b`, từ phải sang trái). Kết quả lưu trong một dòng nhân, tức chuỗi `c`, *từ trái sang phải* (để dễ cộng với chuỗi `d`).

Phép nhân cũng có `carry` (có nhớ) và khi nhân xong cũng cần kiểm tra `carry` cuối cùng để đưa trọn vào chuỗi `c` và tiến hành đóng chuỗi.

- Mỗi dòng nhân `c` tính được sẽ được cộng vào chuỗi `d` (chuỗi chứa tổng kết quả), từ trái sang phải, bắt đầu tại vị trí `i` với dòng nhân thứ `i`.

Phép cộng cũng có `carry` và khi cộng xong cũng cần kiểm tra `carry` cuối cùng, chuỗi `d` chưa đóng ngay (vì còn phải cộng thêm) nhưng phải dùng `k` để ghi nhớ vị trí cuối chuỗi. Khi cộng xong tất cả các dòng nhân, sẽ dùng ký tự null để đóng chuỗi `d` tại vị trí `k` này.

- Khi tính xong một dòng nhân `c` và cộng chúng vào `d`. Dòng nhân `c` sẽ được đảo ngược lại bằng hàm `reverse()` và xuất ra. Để cho kết quả xuất như thật, dùng `l` để

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

cân chỉnh kết quả in ra, `l` giảm `1` đơn vị với mỗi dòng nhân nên kết quả các dòng nhân được xuất dịch dần sang trái.

- Khi đã xuất tất cả các dòng nhân, nghĩa là đã cộng tất cả các dòng nhân vào `d` và đóng chuỗi `d`, tổng kết quả trong chuỗi `d` sẽ được đảo ngược lại bằng hàm `reverse()` và xuất ra.

Nếu không cần in các tính toán trung gian, có thể dùng giải thuật nhân hai số nguyên dài cho kết quả nhanh hơn:

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

char* reverse( char* );     /* đã viết ở trên */

char* multiple( char* a, char* b ) {
  int alen = strlen( a ) - 1;
  int blen = strlen( b ) - 1;
  int n = alen + blen;
  int sum = 0;
  int k, m;
  char* c = ( char* )malloc( n + 1 );

  for ( k = n; k >= 0; --k ) {
    for ( m = k; m >= 0; --m )
      if ( m <= alen && k - m <= blen )
        sum += ( a[m] - 48 ) * ( b[k - m] - 48 );
    c[n - k] = ( sum % 10 ) + 48;
    sum /= 10;
  }
  if ( sum ) c[++n] = sum + 48;
  c[++n] = '\0';
  return reverse( c );
}

int main() {
  char a[] = "87654321";
  char b[] = "12345678";

  printf("%s * %s = %s\n", a, b, multiple( a, b ) );
  return 0;
}
```

Bài 126: (trang 37)

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

void occur( const char *s, char c ) {
  const char* p = s;
  while ( ( p = strchr( p, c ) ) != NULL ) {
    printf( "%u ", p - s );
    p++;
  }
  putchar( '\n' );
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
int main() {
  char s[100], *t, c;

  printf( "Nhap chuoi: " );
  fgets( s, 100, stdin );
  if ( ( t = strrchr( s, '\n' ) ) != NULL ) *t = '\0';
  printf( "Tim ky tu nao? " );
  scanf( "%c", &c );
  printf( "Vi tri xuat hien: " );
  occur( s, c );
  return 0;
}
```

Bài tập này cũng dùng "vòng lặp xử lý chuỗi điển hình" tương tự như bài 116 (trang 186). Tuy nhiên, vì cần tìm vị trí xuất hiện của ký tự nên hàm xử lý chuỗi dùng trong vòng lặp là hàm `strchr()`.

Bài 127: (trang 37)

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

int _isalpha( char c ) {
  return ( c >= 'A' && c <= 'Z' ) || ( c >= 'a' && c <= 'z' );
}

size_t _strlen( const char* s ) {
  const char* p = s;
  while ( *p++ ) { }
  return p - s - 1;
}

char* reverseWord( char* s ) {
  /* con trỏ p1 chỉ cuối chuỗi gốc */
  char *p1 = s + _strlen( s ) - 1;
  /* s1 là chuỗi kết quả, con trỏ q chỉ chuỗi s1 */
  char *s1 = ( char* )calloc( 1, _strlen( s ) + 1 );
  char *q = s1;
  char *p2, *p3;
  p2 = p1;
  /* xử lý từ cuối chuỗi s lên đầu */
  do {
    /* sao chép những ký tự không phải alphabet ở cuối chuỗi s sang s1 */
    while ( p2 >= s && !_isalpha( *p2 ) ) { *q++ = *p2--; }
    /* lấy từng từ, p1 dò tìm đầu từ, p2 giữ đuôi từ */
    p1 = p2;
    if ( p1 >= s ) {
    /* dịch chuyển p1 lên đầu chuỗi để tìm đầu từ */
      while ( p1 > s && _isalpha( *( p1 - 1 ) ) ) p1--;
      /* p3 dùng sao chép từ tìm được sang chuỗi q */
      p3 = p1;
      while ( p3 <= p2 ) { *q++ = *p3++; }
      /* cập nhật lại p2 */
      p2 = p1 - 1;
    }
  } while ( p1 >= s );
  return s1;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
}

int main() {
  char s[] = " cam  khong  duoc  do  rac   ";
  char* s1 = reverseWord( s );

  printf( "Chuoi goc: [%s]\n", s );
  printf( "Chuoi dao: [%s]\n", s1 );
  free( s1 );
  return 0;
}
```

Cách tiến hành đảo từ trong chuỗi đã chú thích chi tiết trong chương trình. Hình dưới minh họa hai thao tác xử lý chính trong hàm `reverseWord()`.

- Sao chép những ký tự không phải alphabet: những ký tự này do con trỏ `p2` chỉ đến, con trỏ `p2` vừa duyệt các ký tự này từ cuối chuỗi ra trước, vừa sao chép chúng sang chuỗi kết quả, tại vị trí con trỏ `q`. Chú ý phát biểu sao chép đặc trưng:

```
*q++ = *p2--;
```

- Sao chép một từ:

Trước hết cần xác định phạm vi một từ: lúc bắt đầu duyệt một từ con trỏ `p2` đã chỉ cuối từ, con trỏ `p1` xuất phát từ `p2` tìm ngược ra trước đến đầu từ. Như vậy `p1` và `p2` xác định phạm vi một từ.

Tiếp theo là sao chép từ: con trỏ `p3` vừa duyệt các ký tự này từ `p1` đến `p2`, vừa sao chép chúng sang chuỗi kết quả, tại vị trí con trỏ `q`, cũng bằng phát biểu sao chép đặc trưng:

```
*q++ = *p3++;
```

Sau đó con trỏ `p2` cập nhật lại vị trí, chỉ đến phần tử không phải alphabet đầu tiên trước từ, hai thao tác trên lại lặp lại. Cần chú ý kiểm tra biên, bảo đảm không vượt quá đầu mảng. Nhắc lại, kiểm tra biên (bounds checking) là công việc quan trọng nhất của lập trình viên C khi làm việc với mảng hoặc chuỗi.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Chuỗi đích s1 được cấp phát bằng calloc(), mục đích là các phần tử đều khởi tạo bằng 0, mặc nhiên có trước phần tử '\0' ở cuối chuỗi.
Với sự hỗ trợ của các hàm xử lý chuỗi trong <string.h>, các thao tác xử lý tiến hành dễ dàng hơn rất nhiều và chỉ cần một con trỏ p duyệt ngược toàn chuỗi gốc.
- Sao chép những ký tự không phải alphabet: con trỏ p vừa duyệt vừa sao chép từng ký tự này vào chuỗi tạm t, sau đó dùng hàm strcat() nối chuỗi tạm t vào cuối chuỗi kết quả s1.
- Sao chép một từ: sau khi vượt qua các ký tự không phải alphabet, con trỏ p tiếp tục di chuyển tới đầu từ kế tiếp, đến ký tự không phải alphabet mới ngay trước từ. Tại đây, dùng hàm strtok() để tách từ ra rồi lại dùng hàm strcat() nối từ tách được vào cuối chuỗi kết quả s1.

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

int main() {
  char s[] = " con  co  con an  con  ca con   ";
  char *s1 = ( char* )calloc( 1, strlen( s ) + 1 );
  char *t = ( char* )calloc( 1, strlen( s ) + 1 );
  int i;
  char *p = s + strlen(s) - 1;   /* con trỏ p chỉ cuối chuỗi gốc */

  printf( "Chuoi goc: [%s]\n", s );
  do {
    i = 0;
    /* sao chép những ký tự không phải alphabet ở cuối chuỗi s sang t */
    while ( p >= s && !isalpha( *(p) ) )
      t[i++] = *p--;
    t[i] ='\0';
    /* dùng strcat nối vào cuỗi chuỗi kết quả s1 */
    strcat( s1, t );
    /* dịch chuyển p lên đầu chuỗi để tìm đầu từ */
    while ( p >= s && isalpha( *(p) ) ) p--;
    if ( p >= s ) {
      p++;
      /* dùng strtok tách từ rồi dùng strcat nối vào cuỗi chuỗi kết quả s1 */
      if ( ( p = strtok( p, t ) ) != NULL )
        strcat( s1, p );
      p--;
    }
  } while( p >= s );
  printf( "Chuoi dao: [%s]\n", s1 );
  free( t );
  free( s1 );
  return 0;
}
```

Bài 128: (trang 37)

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

char* maxW( char *s, int *oldMax ) {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  char *p = strdup( s );
  char *t = NULL;
  while ( ( p = strtok( p, " " ) ) != NULL ) {
    if ( strlen(p) > *oldMax ) *oldMax = strlen( t = p );
    p = NULL;
  }
  return t;
}

void maxWords( char *s ) {
  char *t;
  char *p = s;
  int oldMax = 0;
  while ( ( t = maxW(p, &oldMax ) ) != NULL ) {
    printf( "%s[%d] ", t, oldMax-- );
    p = strstr( p, t ) + strlen( t );
  }
}

int main() {
  char s[255];

  printf( "Nhap chuoi: " );
  scanf( "%254[^\n]s", s );

  maxWords( s );
  putchar( '\n' );
  return 0;
}
```

Hàm `maxW( char* s, int *oldMax )` sẽ trả về từ *dài nhất tìm thấy đầu tiên* trong chuỗi `s`. Kích thước của từ này phải lớn hơn `oldMax` truyền cho hàm, mục tiêu để tìm từ dài nhất (cùng kích thước) kế tiếp. Để thực hiện điều này, `maxW()` dùng `strtok()` để tách các từ và so sánh kích thước với `oldMax`, `oldMax` được cập nhật với từ dài hơn tìm thấy. `oldMax` truyền bằng con trỏ nên thay đổi khi `maxW()` thực thi xong.
Hàm `maxWords()` sử dụng "vòng lặp xử lý chuỗi điển hình". Sau khi gọi `maxW()` với `oldMax` bằng `0` để xuất từ dài nhất tìm thấy đầu tiên, từ `t`, bạn chú ý đến cách vòng lặp xử lý tiếp:
- Giảm `oldMax` đi `1` đơn vị, do `maxW()` tìm các từ dài hơn `oldMax` nên chỉ tìm được các từ có kích thước bằng từ dài nhất tìm thấy đầu tiên.
- Chuỗi để tìm tiếp, tức chuỗi truyền cho `maxW()` bắt đầu ngay sau từ `t`. Ta cập nhật `p` để xác định chuỗi này: dùng `strstr()` tìm từ `t` trong `p` cũ rồi di chuyển `p` đến ngay sau từ `t` này để xác định chuỗi tìm tiếp.

Bài 129: (trang 38)

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

int main() {
  char s[80], *t;
  size_t i, j;

  printf( "Nhap chuoi (chua it nhat 4 so): " );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  fgets( s, 80, stdin );
  if ( ( t = strrchr( s, '\n' ) ) != NULL ) *t = '\0';

  for ( i = 0; s[i]; ++i )
    if ( !isdigit( s[i] ) )
      { strcpy( s + i, s + i + 1 ); i--; }

  if ( strlen( s ) < 4 )
    { printf( "Phai co it nhat 4 so\n" ); return 0; }

  for ( i = 0; i < 4; ++i ) {
    size_t maxpos = i;
    for ( j = i; j < strlen( s ) + i - 3; ++j )
      if ( s[maxpos] < s[j] ) maxpos = j;
    if ( maxpos > i ) strcpy( s + i, s + maxpos );
  }
  s[4] = '\0';
  printf( "So lon nhat con lai: %s\n", s );
  return 0;
}
```

Chuỗi nhận vào được xử lý như sau:

- Đầu tiên loại bỏ tất cả các ký tự không phải số: kiểm tra bằng `isdigit()` của `ctype.h` và xóa bằng `strcpy()` của `string.h`. Sau khi xóa phải lùi con trỏ lại vì ký tự phía sau đã chồng lên vị trí xóa nên có khả năng bị bỏ qua khi vòng lặp duyệt tiếp (xem bài 64, trang 119) .
- Kiểm tra chiều dài chuỗi (chỉ chứa số) còn lại, bảo đảm chiều dài phải > 4.
- Số $\overline{abcd}$ có trị lớn nhất khi $a \geq b \geq c \geq d$. Trên cơ sở này, ta tiến hành tuyển lần lượt các số (một ký tự) lớn nhất, lớn thứ hai, … từ đầu chuỗi đến cuối để tạo chuỗi còn lại có trị lớn nhất.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Chú ý giới hạn cuối của vòng lặp tìm số lớn nhất. Giới hạn này bảo đảm số phần tử chọn được ít nhất là 4 phần tử. Nếu không, sau khi chọn một phần tử, có khả năng không còn đủ số phần tử để chọn nữa.

Bài 130: (trang 38)

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

int PatternSearch( char *s1, char *s2 ) {
  char* pat[100] = { 0 };
  char* p = strdup( s2 );
  int i = 0;
  while( ( p = strtok( p, "*" ) ) != NULL ) {
    pat[i++] = p;
    p = NULL;
  }

  p = s1;
  i = 0;
  while ( pat[i] && ( p = strstr( p, pat[i] ) ) != NULL ) {
    p += strlen( pat[i] );
    i++;
  }
  if ( ( s2[0] != '*' && strstr( s1, pat[0] ) != s1 ) ||
       ( s2[strlen( s2 ) - 1] != '*' &&
         strcmp( s1 + strlen( s1 ) - strlen( pat[i - 1] ), pat[i - 1] ) )
)
    return 0;
  return ( !pat[i] );
}

int main() {
  char* str[] = { "television", "menu", "editions", "education" };
  char* pattern[] = { "*e*u*", "e*i*n", "*e*o*" };
  int i, j, n = 4, m = 3;

  printf( "Danh sach cac tu: " );
  for ( i = 0; i < n; ++i ) printf( "%s ", str[i] );

  for ( j = 0; j < m; ++j ) {
    printf( "\nTim [%s]: ", pattern[j] );
    for ( i = 0; i < n; ++i )
      if ( PatternSearch( str[i], pattern[j] ) )
        printf( "%s ", str[i] );
  }
  putchar( '\n' );
  return 0;
}
```

Đầu tiên ta tách các token trong chuỗi mẫu `s2`, với delimiter là ký tự `'*'`, đặt vào mảng chuỗi `pat`. Sau đó ta dùng hàm `strstr()` để kiểm tra xem các token này có tìm thấy trong chuỗi cần kiểm tra `s1` không, chú ý các điều kiện:

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

- Các token phải được tìm thấy *lần lượt theo thứ tự* trong `s1`: khi tìm thấy token `pat[i]`, tăng con trỏ dò tìm (`p += strlen(pat[i])`) để vượt qua token tìm thấy rồi mới tìm với token kế tiếp `pat[i + 1]`.
- Các token phải *tìm thấy đầy đủ* trong `s1`: xác định yêu cầu này bằng cách sau khi ra khỏi vòng lặp `while` kiểm tra xem `pat[i]` có bằng null hay không. Chú ý mảng chuỗi `pat` khởi tạo các phần tử đều bằng null trước khi nhận token.

Chúng ta cũng cần giải quyết hai trường hợp đặc biệt:
- Không có ký tự `'*'` đầu chuỗi `s2`: ví dụ `[e*i*n]` không so trùng "`television`". Khi đó, con trỏ tìm `pat[0]` (token đầu tiên) trong `s1` *phải trùng* với `s1`.
- Không có ký tự `'*'` cuối chuỗi `s2`: ví dụ `[e*i*n]` không so trùng "`editions`". Khi đó, con trỏ tìm `pat[i - 1]` (token cuối, sau khi ra khỏi vòng `while` kiểm tra) trong `s1` phải trùng với chuỗi con *cuối* `s1` *có kích thước tương tự*.

Bài 131: (trang 38)

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

int isSym( char *s ) {
  char* l = s;
  char* r = s + strlen( s ) - 1;
  do {
    while ( !isalpha( *l ) ) l++;
    while ( !isalpha( *r ) ) r--;
    if ( tolower( *l++ ) != tolower( *r-- ) ) return 0;
  } while ( l < r );
  return 1;
}

void delChar( char *s, char c ) {
  char* p = s;
  while ( ( p = strchr( p, c ) ) != NULL )
    strcpy( p, p + 1 );
  c = isupper( c ) ? tolower( c ) : toupper( c );
  p = s;
  while ( ( p = strchr( p, c ) ) != NULL )
    strcpy( p, p + 1 );
}

int main() {
  char s[100], c, *t;

  printf( "Nhap chuoi: " );
  fgets( s, 100, stdin );
  if ( ( t = strrchr( s, '\n' ) ) != NULL ) *t = '\0';
  if ( isSym( s ) ) printf( "Chuoi doi xung\n" );
  else printf( "Chuoi khong doi xung\n" );
  printf( "Xoa ky tu nao? " );
  scanf( "%c", &c );
  delChar( s, c );
  printf( "%s\n", s );
  return 0;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
}
```

Kiểm tra chuỗi đối xứng không thể dựa vào chỉ số các phần tử trong chuỗi, như so sánh cặp `s[i]` với `s[strlen(s) - i - 1]`, vì trong chuỗi còn những ký tự không phải alphabet, ta cần bỏ qua không quan tâm đến các ký tự này.

Trong vòng lặp kiểm tra `do while`:

- Vòng lặp con `while` với biến đếm `l` dùng vượt qua các ký tự *không* phải alphabet nếu có, kể từ đầu mảng.
- Vòng lặp con `while` với biến đếm `r` dùng vượt qua các ký tự *không* phải alphabet nếu có, kể từ cuối mảng.

Khi cả hai vòng lặp con trên chấm dứt:

- Nếu `l < r`, nghĩa là `r` chỉ một ký tự alphabet đầu chuỗi, `l` chỉ một ký tự alphabet cuối chuỗi, "đối xứng" với `r` (theo nghĩa không quan tâm đến các ký tự không phải alphabet). Ta so sánh chúng ở dạng chữ thường. Nếu hai ký tự này giống nhau, tăng `l` và giảm `r` để kiểm tra tiếp; ngược lại trả về False.
- Vòng lặp `do while` chấm dứt khi `l ≥ r`, quá trình kiểm tra thành công, trả về True.

Để xóa ký tự chỉ định, ta dùng "vòng lặp xử lý chuỗi điển hình" hai lần: một lần dùng xóa ký tự chữ thường, một lần dùng xóa ký tự chữ hoa hoặc ngược lại.

Bài tập: Nhập một chuỗi ký tự:

- Xuất vị trí và đếm các ký tự là nguyên âm trong chuỗi.
- Xuất và đếm các chuỗi con đối xứng có 5 ký tự.

```
Nhap chuoi: nhuthuhuhu ↵
[2] [5] [7] [9]
Co 4 nguyen am
huhuh
uhuhu
Co 2 chuoi 5 ky tu doi xung
```

```c
#include <stdio.h>
#include <string.h>
#define MAX 100

int main() {
  char s[MAX];
  char s1[6] = { 0 };
  char *p = s;
  int i, n, c1 = 0, c2 = 0;

  printf( "Nhap chuoi: " );
  scanf("%99[^\n]s", s);

  for ( ;*p; p++ )
    if ( strchr( "aeiou", *p ) != NULL && ++c1 )
      printf("[%u] ", p - s );
  printf( "\nCo %d nguyen am\n", c1 );
  n = strlen( s ) - 4;

  for ( i = 0; i < n; ++i )
    if ( s[i] == s[i + 4] && s[i + 1] == s[i + 3] && ++c2 )
      printf( "%s\n", strncpy( s1, s + i, 5 ) );
  printf( "Co %d chuoi 5 ky tu doi xung\n", c2 );
  return 0;
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 132: (trang 38)

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

int main() {
  char a[80], b[80];

  printf( "Nhap chuoi a: " );
  fgets( a, 80, stdin );
  printf( "Nhap chuoi b: " );
  fgets( b, 80, stdin );
  if ( strtok( a, b ) == NULL && strtok( b, a ) == NULL )
    printf( "Tao tu cung cac ky tu\n" );
  else
    printf( "Co ky tu khac nhau\n" );
  return 0;
}
```

Nhắc lại, tham số thứ hai của hàm `strtok()` là một chuỗi chứa các ký tự ngăn cách (delimiter) giữa các token mà `strtok()` cần phải tách ra từ chuỗi là tham số thứ nhất. Như vậy:

- Nếu `strtok( a, b ) == NULL`, nghĩa là `a` chỉ chứa các ký tự của `b`, vì `a` chỉ chứa toàn các ký tự ngăn cách nên không tách được token.
- Tương tự, nếu `strtok( b, a ) == NULL`, nghĩa là `b` chỉ chứa các ký tự của `a`.

Nếu cả hai trường hợp trên cùng xảy ra, `a` và `b` được tạo ra với cùng tập ký tự.

Bài 133: (trang 38)

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

int convert( char c ) {
  if ( !isxdigit( c ) ) return -1;
  if ( isdigit( c ) ) return c - '0';
  if ( islower( c ) ) return c - 'a' + 10;
  return c - 'A' + 10;
}

unsigned long hextoulong( char* s ) {
  int i;
  unsigned long n = 0;
  for ( i = 0; s[i]; ++i ) {
    int t = convert( s[i] );
    if ( t == -1 ) return t;
    n =  n * 16 + t;
  }
  return n;
}

int main() {
  char s[10], *t;
  unsigned long n;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  printf( "Nhap chuoi hex: " );
  fgets( s, 10, stdin );
  if ( ( t = strrchr( s, '\n' ) ) != NULL ) *t = '\0';

  if ( ( n = hextoulong( s ) ) == -1 )
    printf( "Chuoi chua ky tu khong hop le\n" );
  else
    printf( "Decimal: %lu\n", n );
  return 0;
}
```

Phương pháp Horner trình bày trong phần hướng dẫn, tổng quát như sau:
`d = P(16) = (...((s[0].16 + s[1]).16 + s[2]).16 + ... + s[n − 2]).16 + s[n − 1]` xem bài 122 (trang 194, nhị phân → thập phân) và 124 (trang 197, chuỗi số → thập phân)
Khi tính toán, hàm `convert()` sẽ ánh xạ từng ký tự hex thành trị thập phân cụ thể.

Bài 134: (trang 39)

```
#include <stdio.h>

int convert( char* s, int i ) {
  char c = s[i];
  if ( c == 'M' ) return 1000;
  if ( c == 'D' ) return 500;
  if ( c == 'C' ) return 100;
  if ( c == 'L' ) return 50;
  if ( c == 'X' ) return 10;
  if ( c == 'V' ) return 5;
  if ( c == 'I' ) return 1;
  return 0;
}

int rtoi( char* s ) {
  int n, a, i;
  a = n = convert( s, 0 );
  for ( i = 1; s[i]; ++i ) {
    int b = convert( s, i );
    if ( 10 * a < b ) return 0;
    n += b;
    if ( b > a ) n -= 2 * a;
    a = b;
  }
  return n;
}

int main() {
  int n;

  printf( "MCMXCIX = " );
  if ( ( n = rtoi( "MCMXCIX" ) ) == 0 )
    printf( "So khong hop le\n" );
  else
    printf( "%d\n", n );
  printf( "MCMIC = " );
  if ( ( n = rtoi( "MCMIC" ) ) == 0 )
    printf( "So khong hop le\n" );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  else
    printf( "%d\n", n );
  return 0;
}
```

Hàm `convert()` sẽ ánh xạ ký tự `s[i]` thành trị thập phân cụ thể.
Trên luật tạo số La mã, với cặp ký tự La mã $\overline{ab}$:
- Nếu `a` ≥ `b`, trị thập phân của chúng sẽ là (trị của `a` + trị của `b`).
- Nếu `a` < `b`:
Nếu `10 * a < b`, sai luật tạo số La mã, hàm trả về `0` (hệ thống số La mã không có số `0`). Ngược lại, trị thập phân của chúng sẽ là (trị của `b` - trị của `a`). Vì trị của `a` đã được cộng dồn vào tổng chung trước khi so sánh nên ta dùng công thức tương đương sau: (trị của `a` + trị của `b`) - 2 * (trị của `a`)

Bài 135: (trang 39)

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

char* strrepl( char* s, char* pat, char* rep ) {
  size_t plen, rlen, d = 0;
  char *t, *p, *q = s;
  if ( !strcmp( pat, rep ) )
    return strdup( s );
  plen = strlen( pat );
  rlen = strlen( rep );
  while ( ( p = strstr( q + d, pat ) ) != NULL ) {
    d = p - q + rlen;
    t = ( char * )calloc( strlen( q ) + rlen - plen + 1, sizeof( char ) );
    if ( !t ) return NULL;
    strncpy( t, q, p - q );
    strncpy( t + ( p - q ), rep, rlen );
    strcpy( t + ( p - q ) + rlen, p + plen );
    free( q );
    q = t;
  }
  return d ? q : s;
}

int main() {
  char s[] = "Ta mo thay em o mot noi em xa lam";
  char pat[] = "em";
  char rep[] = "em yeu";
  char* t = strrepl( s, pat, rep );

  printf( "Chuoi goc: %s\n", s );
  printf( "Thay the '%s' voi '%s'\n", pat, rep );
  printf( "Chuoi moi: %s\n", t );
  return 0;
}
```

Chú ý chuỗi thay thế có thể:
- chứa chuỗi tìm kiếm.
- giống với chuỗi tìm kiếm.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

- dài hoặc ngắn hơn chuỗi tìm kiếm.
- chuỗi rỗng.

Hàm `strrepl( char* s, char* pat, char* rep )`, dùng thay thế tất cả chuỗi con `pat` trong chuỗi `s` bằng chuỗi con `rep`, hoạt động như sau:

- Trả về `NULL` trong trường hợp không có thay thế nào xảy ra: `rep` giống `pat` hoặc không có `pat` trong `s`.
- Tìm vị trí cần thay thế `p` bằng hàm `strstr()`.
- Cấp phát một chuỗi `q` mới, vừa đủ chứa chuỗi kết quả.
- Lần lượt sao chép: đoạn trước chuỗi thay thế, chuỗi thay thế, đoạn sau chuỗi thay thế; tạo thành chuỗi kết quả. Chú ý dùng hàm xử lý chuỗi thích hợp và tính toán chính xác vị trí bắt đầu của nơi sẽ sao chép chuỗi đến.

Bài 136: (trang 39)

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

int main() {
  char s[100];
  char delimiter[] = " \t\"'?!.,:;";
  char freq[26] = { 0 };
  char *p;
  int lines, words, i;

  lines = words = 0;
  while ( fgets( s, 100, stdin ) != NULL ) {
    char *t = strrchr( s, '\n' );
    if ( t != NULL ) *t = '\0';
    lines++;
    if ( strcmp( s, "") == 0 ) continue;

    for ( i = 0; s[i]; ++i )
      if ( isalpha( s[i]) ) freq[toupper(s[i]) - 'A']++;

    p = s;
    while ( ( p = strtok( p, delimiter ) ) != NULL )
      { words++; p = NULL; }
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  }
  printf( "%d hang, %d tu, voi tan so cac ky tu:\n", lines, words );
  for ( i = 0; i < 26; ++i ) {
    printf( "%c: %d\t", i + 'A', freq[i] );
    if ( i > 0 && ( i + 1 )%7 == 0 ) putchar( '\n' );
  }
  putchar( '\n' );
  return 0;
}
```

Ta xử lý mỗi dòng nhập như sau:
- Xử lý sơ bộ: xóa ký tự `'\n'` do `fgets()` lưu và kiểm tra dòng rỗng.
- Đếm dòng: với mỗi vòng lặp, nghĩa là với mỗi lần `fgets()` hoạt động thành công, tăng biến đếm dòng `lines`.
- Đếm tần số xuất hiện các chữ cái: dùng mảng `freq[26]` (cho 26 chữ cái tiếng Anh). Duyệt chuỗi, với mỗi chữ cái tăng tần số của nó tại phần tử tương ứng trong mảng `freq`.
- Đếm từ: dùng "vòng lặp xử lý chuỗi điển hình" với `strtok()`, tách từng từ và tăng biến đếm từ `words`.

Bài 137: (trang 39)

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

void allTrim( char* s ) {
  char* p;
  while ( ( p = strstr( s, "  " ) ) != NULL )
    strcpy( p, p + 1 );
  if ( *s == ' ' ) strcpy( s, s + 1 );
  p = s + strlen( s ) - 1;
  if ( *p == ' ' ) *p = '\0';
}

int main() {
  char s[30];
  char sname[4] = " x.";
  char *p, *out;
  char name[100][30];
  int line, i;

  line = 0;
  while ( fgets( s, 30, stdin ) != NULL ) {
    char* t = strrchr( s, '\n' );
    if ( t != NULL ) *t = '\0';
    if ( strcmp( s, "" ) == 0 ) continue;
    allTrim( s );
    /* lấy từ cuối của chuỗi, t lưu vị trí từ này */
    p = strrchr( s, ' ' );
    t = p + 1;
    out = strdup( t );
    strcat( out, ", " );
    /* lấy từ đầu bằng strtok(s), trọn từ */
    if ( ( p = strtok( s, " " ) ) != NULL )
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
      strcat( out, p );
    /* lấy các từ sau bằng strtok(NULL), chuyển thành dạng viết tắt */
    while ( ( p = strtok( NULL, " " ) ) != t ) {
      sname[1] = p[0];  /* ký tự x trong chuỗi sname thành tên viết tắt */
      strcat( out, sname );
    }
    /* lưu vào mảng chuỗi name */
    strcpy( name[line++], out );
  }
  for ( i = 0; i < line; ++i )
    printf( "%s\n", name[i] );
  return 0;
}
```

Sơ đồ "tháo lắp" một chuỗi được mô tả trong hình dưới:

Bài 138: (trang 40)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 50

int main() {
  char s[MAX + 1];
  char* lines[100];
  int i, n = 0;

  while ( fgets( s, MAX, stdin ) != NULL ) {
    char* t = strrchr( s, '\n' );
    if ( t != NULL ) *t = '\0';
    if ( strcmp( s, "" ) == 0 ) continue;
    lines[n] = ( char* )malloc( MAX + 1 );
    snprintf( lines[n], MAX - ( t - s ) + strlen( s ) + 1,
              "%*c%s", MAX - ( t - s ), ' ', s );
    n++;
  }
  for ( i = 0; i < n; ++i ) {
    printf( "%s\n", lines[i] );
    free( lines[i] );
  }
  return 0;
}
```

Hàm `sprintf(char *buf, const char *format, ...);` hoạt động tương tự `printf()`, nhưng nó ghi kết quả ra một chuỗi `buf` (có ký tự null cuối chuỗi), chứ không xuất ra

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

thiết bị xuất chuẩn (stdout). Như vậy, hàm sprintf() cung cấp cho chúng ta một cách "lắp ráp" chuỗi theo ý muốn. Tham số thứ nhất của sprintf() là địa chỉ của chuỗi đích, các tham số còn lại tương tự printf().

Tuy nhiên, hàm này được khuyến cáo là không an toàn do các tham số có thể làm tràn buf. Thay thế bằng:

```
snprintf(char *buf, size_t n, const char *format, ...);
```

Về chuỗi định dạng "%*", xem bài 28 (trang 90).

Bài 139: (trang 40)

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

char* pack( char* s ) {
  char *s1, *p, *t;
  size_t d;
  s1 = strdup( s );
  p = t = s;
  d = 0;
  do {
    while ( *p == *t ) t++;
    if ( t - p > 1 ) {
      char buf[5];
      itoa( t - p, buf, 10 );
      d += snprintf( s1 + d, strlen( buf ) + 2, "%s%c", buf, *p );
    } else
      d += snprintf( s1 + d, 2, "%c", *p );
    p = t;
  } while ( *t );
  return s1;
}

char* unpack( char* s ) {
  char *s1;
  size_t d, i;
  for ( d = i = 0; s[i]; ++i )
    if ( isdigit( s[i] ) ) {
      int k = ( s[i] - '0' );
      while ( isdigit( s[++i] ) )
        k = ( k * 10 ) + ( s[i] - '0' );
      d += k;
    } else d++;
  s1 = ( char* )calloc( d + 1, sizeof( char ) );

  for ( d = i = 0; s[i]; ++i )
    if ( isdigit( s[i] ) ) {
      int k = ( s[i] - '0' );
      while ( isdigit( s[++i] ) )
        k = ( k * 10 ) + ( s[i] - '0' );
      memset( s1 + d, s[i], k );
      d += k;
    } else d += snprintf( s1 + d, 1, "%c", s[i] );
  return s1;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
}

int main() {
  char s[] = "aaabccccddddeeeeeeeeeeeeefghhhhhhiiiiaaaabbbbbbc";
  char* t;

  printf( "Chuoi goc: %s\n", s );
  t = pack( s );
  printf( "Nen      : %s [%.1f%%]\n", t, strlen( t ) * 100.0 / strlen( s ) );
  printf( "Giai nen : %s\n", t = unpack( t ) );
  return 0;
}
```

Hàm `snprintf()` (bài 138, trang 214) trả về một số `int` là chiều dài của chuỗi do hàm tạo thành (không tính ký tự null).
Hàm `void *memset(void *buf, int ch, size_t count);` sao chép trị `ch` (chuyển kiểu thành `unsigned char`) đến `count` byte đầu tiên của vùng nhớ chỉ bởi `buf`.
Hàm `itoa(int value, char* buf, int radix);` chuyển trị `value` dưới hệ số `radix` thành chuỗi `buf`.
Mỗi cặp nén `na` bao gồm phần số `n` (`n > 1`, có thể có nhiều ký tự) và ký tự `a`.
Hình dưới mô tả cách nén và giải nén "run":

- Nén "run": (hình bên trái) hiệu của hai con trỏ `p` (chỉ đầu "run") và `t` (dùng trong vòng lặp để dò tìm cuối "run") xác định chiều dài của "run", cũng là phần số trong cặp nén. Ký tự trong cặp nén chính là nội dung của phần tử do `p` trỏ đến. Cặp nén được hàm `snprintf()` đặt vào chuỗi kết quả `s1` tại vị trí `d` (tức `s1 + d`). Trị `d` là kết quả trả về của hàm `snprintf()`, trị `d` được khởi tạo bằng `0` (đầu chuỗi `s1`) và sẽ được cập nhật liên tục để xác định chính xác nơi đặt cặp nén kế tiếp vào.
- Giải nén "run": (hình bên phải) phần số trong cặp nén được lấy ra và chuyển thành số trước. Nó được dùng như tham số thứ ba của hàm `memset()` để xác định số byte sẽ được thiết lập bởi `memset()`, tức chiều dài "run". Ký tự kế tiếp được dùng như tham số thứ hai của hàm `memset()`, là ký tự được thiết lập cho toàn bộ "run". Hàm `memset()` bắt đầu hoạt động từ `s1 + d`. Trị `d` khởi tạo bằng `0` và sẽ được cập nhật liên tục để xác định chính xác nơi hàm `memset()` bắt đầu hoạt động.
Với "run" một ký tự không nén, giải thuật đơn giản nên không mô tả ở đây.
Trước khi giải nén, chuỗi nén được duyệt để xác định chiều dài chuỗi giải nén, giúp cấp phát chính xác.

Bài 140: (trang 40)

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

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
int IsISBN( char* ISBN ) {
  int i, k, c, sum;

  k = 10;
  for ( sum = i = 0; ISBN[i]; ++i )
    if ( isdigit( ISBN[i] ) || toupper( ISBN[i] ) == 'X' ) {
      c = ( toupper( ISBN[i] ) == 'X' ) ? 10 : ISBN[i] - '0';
      sum += c * k;
      k--;
    }
  return ( sum % 11  == 0 );
}

int main() {
  char* s = "0-13-362658-X";
  if ( IsISBN( s ) ) printf( "ISBN %s hop le\n", s );
  else               printf( "ISBN %s khong hop le\n", s );
  return 0;
}
```

Xem chuỗi ISBN như mảng các ký tự, các ký tự tách ra được kiểm tra bằng hàm `isdigit()` hoặc so sánh với `'X'` để chắc là các ký tự hợp lệ của ISBN, loại các ký tự nối hyphen. Ký tự số được chuyển thành số (trừ cho `'0'`), ký tự `'X'` được chuyển thành `10`. Sau đó thực hiện theo thuật toán đã giới thiệu.

Còn một thuật toán khác dùng kiểm tra số ISBN: lấy từng số của ISBN (trừ số cuối, gọi là check digit) nhân với số thứ tự chỉ vị trí của nó (bắt đầu từ 1, tính từ trái sang phải, không tính dấu nối –). Số dư của phép chia tổng các tích nhận được cho 11 nếu bằng check digit thì số ISBN được kiểm tra là hợp lệ. Ví dụ:

ISBN	0	–	1	3	1	-	1	0	3	7	0	–	9
Vị trí	1		2	3	4		5	6	7	8	9		Không tính
Tích	0 + 2 + 9 + 4 + 5 + 0 + 21 + 56 + 0 = 97												

97 chia cho 11 dư 9, bằng check digit, vậy số ISBN trên hợp lệ.

```
int IsISBN( char* ISBN ) {
  int i, k, c, sum, len;
  len = strlen( ISBN );
  k = 1;
  for ( sum = i = 0; i < len - 1; ++i )
    if ( isdigit( ISBN[i] ) ) {
      sum += k * ( ISBN[i] - '0' );
      k++;
    }
  c = ( toupper( ISBN[len - 1] ) == 'X' ) ? 10 : ISBN[len - 1] - '0';
  return ( sum % 11  == c );
}
```

Bài 141: (trang 41)

```
#include <stdio.h>

double population( int year ) {
  if ( year == 2000 ) return 8E + 9;
  return population( year - 1 ) * ( 1 + 0.025 );
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
int main() {
  printf( "%.f nguoi\n", population( 2010 ) );
  return 0;
}
```

Chúng ta có thể dùng cách tiếp cận như sau để viết một hàm đệ quy:
1. Định nghĩa chính xác vấn đề cần giải quyết. Tốt nhất là trình bày nó dưới dạng một biểu thức truy hồi.
2. Xác định quy mô (kích thước, size) của vấn đề. Kích thước này sẽ được truyền như tham số đến hàm và sẽ thay đổi (thường là thu nhỏ) sau mỗi lần gọi hàm.
3. Xác định và giải quyết trường hợp cơ sở, nghĩa là trường hợp mà vấn đề có thể được giải quyết *không đệ quy*, thường là điều kiện đầu (initial condition) của biểu thức truy hồi. Đây chính là điểm dừng đệ quy, nếu không định nghĩa, hàm đệ quy sẽ thực hiện đến khi tràn stack.
4. Xác định và giải quyết trường hợp tổng quát, thường dựa vào *biểu thức truy hồi* hoặc *tính chất đệ quy* của vấn đề. Giải quyết trường hợp tổng quát theo cách đệ quy thường là thu nhỏ quy mô vấn đề để *chuyển dần nó về trường hợp cơ sở*.
Ví dụ cho bài tập trên:
1. Biểu thức truy hồi:

$$P_y = \begin{cases} 8.10^9 & y = 2000 \\ P_{y-1} + \frac{2.5}{100} P_{y-1} & \end{cases}$$

2. Kích thước đệ quy: năm (year), sẽ giảm cho mỗi lần gọi đệ quy.
3. Trường hợp cơ sở: giải quyết dựa vào điều kiện đầu $P_{2000} = 8.10^9$
4. Trường hợp tổng quát: giải quyết dựa vào biểu thức truy hồi $P_y = P_{y-1}(1 + 2.5\%)$
Từ phân tích trên bạn dễ dàng viết được hàm đệ quy giải quyết vấn đề.
Ngoài ra, chú ý:
- Cách biểu diễn số theo dạng thức khoa học: $aE \pm n$, tương đương với $a.10^{\pm n}$.
- Chuỗi định dạng "%.f" (có dấu .) dùng in số thực không có phần thập phân.

Bài 142: (trang 41)

```
#include <stdio.h>

int main() {
  char c;

  if ( ( c = getchar() ) != '\n' ) main();
  putchar( c );
  return 0;
}
```

Chuỗi đệ quy là một chuỗi gọi hàm liên tục, biến cục bộ cũng như địa chỉ trở về sẽ được lưu vào stack trước khi gọi hàm. Đệ quy thông thường chậm do stack overhead.
Phân biệt hai loại đệ quy:
- Đệ quy đầu:

```
void hello( int n ) {
  hello( n - 1 );          /* gọi đệ quy */
  printf( "%d", n );       /* tác vụ trong một lần đệ quy */
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Lời gọi đệ quy được gọi trước các tác vụ trong một lần đệ quy. Khi kết thúc đệ quy, biến cục bộ được đẩy ra khỏi stack theo nguyên tắc LIFO (Last In First Out - vào sau ra trước) và được sử dụng nên kết quả xuất ngược với thứ tự dùng các biến cục bộ.

- Đệ quy đuôi (tail recursion):

```
void hello( int n ) {
  printf( "%d", n );          /* thực hiện công việc trong một lần đệ quy */
  hello( n - 1 );             /* gọi đệ quy */
}
```

Lời gọi đệ quy được thực hiện cuối cùng. Không có tác vụ nào được thực hiện sau lời gọi đệ quy.

Ta dùng đặc điểm đệ quy đầu này để thực hiện đệ quy với hàm `main()`. Kết quả là các ký tự nhận vào trong mỗi lần đệ quy sẽ được xuất ra với thứ tự ngược lại. Bạn có thể dùng chức năng Call Stack thường có trong tiện ích debug để theo dõi quá trình này.

Bài 143: (trang 41)

```
#include <stdio.h>

int A( int n, int m, int *c ) {
  ( *c )++;
  if ( !n  ) return m + 1;
  if ( !m  ) return A( n - 1, 1, c );
  return A( n - 1, A( n, m - 1, c ), c );
}

int main() {
  int c = 0;

  printf( "A( 3, 6 ) = %d\n", A( 3, 6, &c ) );
  printf( "Goi de quy %d lan\n", c );
  return 0;
}
```

Hàm Ackermann là ví dụ về hàm đệ quy không cơ bản, có trị tăng cực nhanh. Ví dụ dạng thập phân của `A(4, 3)` không trình bày được vì có số chữ số lớn hơn số nguyên tử ước lượng của vũ trụ.

Hàm Ackermann được hiện thực bằng cách dùng biểu thức truy hồi sau:

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

$$A(n, m) = \begin{cases} m + 1 & n = 0 \\ A(n - 1, 1) & m = 0 \\ A(n - 1, A(n, m - 1)) & n, m > 0 \end{cases}$$

Điều kiện đầu: `A(0, m) = m + 1`
Tham số `c` được đưa vào để đếm số lần đệ quy.
`A(3, 6)` gọi đệ quy 172233, lồng sâu 511 cấp, thường dùng đo tốc độ hoạt động (benchmark), ví dụ:

```
#include <time.h>
/* ... */
clock_t t1, t2;
t1 = clock();
A( 3, 6 );
t2 = clock();
printf( "Time taken = %.3f Seconds\n", ( t2 - t1 ) / (double)CLOCKS_PER_SEC );
```

Công thức `A(3, n) =` $2^{n+3}-3$ dùng để tính nhanh `A(3, n)`.
Trong lời gọi đệ quy có truyền tham chiếu bằng con trỏ, tham số `c` của hàm trên phải được truyền như sau: `&(*c)`, nhưng điều này tương đương với `c`.

Bài 144: (trang 41)

```
#include <stdio.h>
#include <math.h>
#define eps 1e - 3

double Pi() {
  static k = 0;
  double E = 4.0 / ( 2 * k + 1 );
  double s = pow( -1, k );
  if ( E < eps ) return 0;
  k++;
  return s * E + Pi();
}

int main() {
  printf( "Pi = %.3lf\n", Pi( 0 ) );
  return 0;
}
```

Biểu thức tính Pi: $\pi = \sum_{k=0} (-1)^k \frac{4}{2k + 1}$ thuận lợi cho việc thực hiện bằng vòng lặp hoặc chuyển sang dùng đệ quy:

```
double Pi( int k ) {
  double E = 4.0 / ( 2 * k + 1 );
  double s = pow( -1, k );
  if ( E < eps ) return 0;
  return s * E + Pi( k + 1 );
}
```

Trong đó `k` (kích thước đệ quy) được thay đổi bằng cách truyền như tham số của những lần gọi đệ quy.
Do yêu cầu của bài tập, hàm `Pi()` được viết *không tham số*, ta dùng biến `static` để lưu lại trị của `k` trong lần gọi trước.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Chú ý là hàm chỉ được gọi một lần từ hàm `main()` của chương trình, vì trị `static k` khởi đầu sẽ thay đổi, ảnh hưởng đến lần gọi sau. Cách thiết kế với biến `static` trong hàm như trên chỉ đáp ứng yêu cầu bài tập, không nên dùng trong thực tế, ví dụ thiết kế các hàm trong thư viện riêng.

Bài 145: (trang 41)

```
#include <stdio.h>

void dec2bin( int x ) {
  if ( x >= 2 ) dec2bin( x / 2 );
  printf( "%d", x % 2 );
}

void dec2hex( int x ) {
  int c = x % 16;
  if ( x >= 16 ) dec2hex( x / 16 );
  printf( "%c", c > 9 ? c - 10 + 'A' : c + '0' );
}

int main() {
  int x;

  printf( "Nhap x: " );
  scanf( "%d", &x );

  printf( "Bin: " );
  dec2bin( x );

  printf( "\nHex: " );
  dec2hex( x );
  putchar( '\n' );
  return 0;
}
```

Chuyển đổi hệ số của số `x`, ví dụ từ hệ thập phân sang hệ thập lục phân, được thực hiện như hình bên:

Chia liên tục `x` cho `16` đến khi thương số bằng `0`, các số dư của phép chia cho `16` được lần lượt lưu vào stack.

Khi phép chia cho `16` chấm dứt (thương số bằng `0`), lần lượt đẩy các số dư ra khỏi stack (theo nguyên tắc `LIFO` - vào sau ra trước) để có kết quả chuyển đổi.

Các trị dư từ `10 - 15` được chuyển thành từ `A - F`.

Vòng lặp chia `x` cho `16` được chuyển thành đệ quy nhằm sử dụng stack hoạt động được tạo ra trong quá trình đệ quy lưu các *số dư* cho phép chia hệ số `16`. Để dữ liệu đưa vào stack đúng thứ tự, ta dùng *đệ quy đầu*. Trong đó, số dư của `phép` chia cho hệ số được lưu vào stack của đệ quy.

Bài tập: Viết hàm `convert(n, k)` đệ quy, trong đó `n` là số thập phân cần chuyển đổi và `k` là hệ số (2, 8, 16).

```
void convert( unsigned n, unsigned k ) {
  if ( n >= k ) convert( n / k, k );
  printf( "%c", ( n % k )[ "0123456789ABCDEF" ] );
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Nhắc lại, trong C, `s[3]` và `3[s]` được xem chỉ là một địa chỉ `s + 3`. Vì vậy:

```
printf( "%c", ( n % k )[ "0123456789ABCDEF" ] );
```

Chỉ là cách viết tắt của:

```
char s[] = "0123456789ABCDEF";
printf( "%c", s[n % k] );
```

Bài 146: (trang 41)

```
#include <stdio.h>

int coef( int n, int k ) {
  if ( k == 0 || k == n ) return 1;
  return coef( n - 1, k - 1 ) + coef( n - 1, k );
}

void tpascal( int n ) {
  int i, j;
  for ( i = 0; i <= n; ++i ) {
    printf( "%*c", ( n - i + 1 ) * 2, ' ' );
    for ( j = 0; j <= i; ++j )
      printf( "%2d%2c", coef( i, j ), ' ' );
    putchar( '\n' );
  }
}

int main() {
  int n;

  printf( "Nhap n (n < 15): " );
  scanf( "%d", &n );
  tpascal( n );
  return 0;
}
```

Dễ dàng tính một hệ số trong tam giác Pascal bằng đệ quy dựa vào biểu thức truy hồi: $C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$ (giải thuật "chia để trị")

Với điều kiện đầu: $C_n^0 = C_n^n = 1$

Tuy nhiên, giải thuật đệ quy này kém hiệu quả. Thực tế, người ta thường dùng giải thuật quy hoạch động:

```
int** coefArray( int n ) {
  int i, j;
  /* cấp phát động mảng B[n + 1][n + 1] */
  int** B = ( int** )calloc( ( n + 1 ), sizeof( int* ) );
  B[0] = ( int* )calloc( ( n + 1)*( n + 1 ), sizeof( int ) );
  for ( i = 0; i < n + 1; ++i )
    B[i] = B[0] + i * ( k + 1 );
  /* quy hoạch động */
  for ( i = 0; i <= n; ++i )
    for ( j = 0; j <= i; ++j )
      if ( j == 0 || j == i ) B[i][j] = 1;
      else B[i][j] = B[i - 1][j - 1] + B[i - 1][j];
  return B;
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
void tpascal( int n ) {
  int i, j;
  int** B = coefArray( n );
  for ( i = 0; i <= n; ++i ) {
    printf( "%*c", ( n - i + 1 ) * 2, ' ' );
    for ( j = 0; j <= i; ++j )
      printf( "%2d%2c", B[i][j], ' ' );
    putchar( '\n' );
  }
  /* giải phóng vùng nhớ được cấp phát */
  free( B[0] );
  free( B );
}
```

Chú ý cách trình bày kết quả xuất của tam giác Pascal, dùng `%*c`.

Bài 147: (trang 42)

```
#include <stdio.h>

void Swap( int *a, int *b ) {
  int t = *a; *a = *b; *b = t;
}

int Max( int* a, int n ) {
  int i, maxpos = 0;
  for ( i = 1; i < n; ++i )
    if ( a[i] > a[maxpos] ) maxpos = i;
  return maxpos;
}

void selectSort( int a[], int n ) {
  if ( n < 2 ) return;
  Swap( a + n - 1, a + Max( a, n ) );
  selectSort( a, n - 1 );
}

int main() {
  int a[] = { 3, 5, 4, 6, 7, 1, 2 };
  int size = sizeof a / sizeof *a;
  int i;

  printf( "Mang goc : " );
  for ( i = 0; i < size; ++i )
    printf( "%d ", a[i] );
  putchar( '\n' );

  selectSort( a, size );
  printf( "Mang tang: " );
  for ( i = 0; i < size; ++i )
    printf( "%d ", a[i] );
  putchar( '\n' );
  return 0;
}
```

Phân tích:

1. Sắp xếp tăng một mảng `A` theo thuật toán sắp xếp kiểu chọn (Selection Sort).

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

2. Kích thước đệ quy: số phần tử của mảng `A`.
3. Trường hợp cơ sở: khi mảng `A` có duy nhất một phần tử, xem như đã sắp xếp, dừng đệ quy.
4. Trường hợp tổng quát: dựa vào cấu trúc đệ quy của mảng (mảng con của một mảng cũng là một mảng), thao tác sắp xếp mảng `A` được mô tả đệ quy như sau:

`Sort(A, n)`	=	`Sort(A, n - 1)`	+	`A[n-1]`
Sắp xếp mảng `A`	=	Sắp xếp mảng con từ `A[0] -> A[n-2]`		Phần tử cuối `A[n-1]` đã sắp xếp

Dấu + biểu thị `A[n-1]` nằm phía sau mảng con.
Thao tác sắp xếp tăng đúng *phần tử cuối* của một mảng (con) theo kiểu *chọn* như sau: hoán chuyển phần tử *lớn nhất* của mảng (con), được tìm bằng hàm `Max()` với phần tử cuối mảng (con).
Hàm `Swap()`, vì có tham số truyền bằng con trỏ, được gọi đầy đủ như sau:

```
Swap( &a[n - 1], &a[Max( a, n )] );
```

Tương đương:

```
Swap( a + n - 1, a + Max( a, n ) );
```

Cách sắp xếp trên là "hoán chuyển phần tử *lớn nhất* của mảng con (từ phần tử đầu đến kế cuối) cho *phần tử cuối mảng*", có tư duy hơi "ngược" với thuật toán cơ bản. Lý do chúng ta muốn giữ chỉ số của phần tử đầu mảng con luôn bằng `0`.
Bạn vẫn có thể làm theo thuật toán cơ bản: "hoán chuyển phần tử *nhỏ nhất* của mảng con (từ phần tử thứ hai đến cuối) cho *phần tử đầu mảng*", nếu nắm vững quan hệ giữa con trỏ và mảng:

```
int Min( int* a, int n ) {
  int i, minpos = 0;
  for ( i = 1; i < n; ++i )
    if ( a[i] < a[minpos] ) minpos = i;
  return minpos;
}

void selectSort( int a[], int n ) {
  if ( n < 2 ) return;
  Swap( a, a + Min( a, n ) );
  a++;
  selectSort( a, n - 1 );
}
```

Hàm `Max()` cũng có thể viết đệ quy, xem bài 151 (trang 228).

Bài 148: (trang 42)

```
#include <stdio.h>

void Swap( int *a, int *b ) {
  int t = *a; *a = *b; *b = t;
}

void bubbleSort( int a[], int n ) {
  int i;
  if ( n < 2 ) return;
  for ( i = 0; i < n - 1; ++i )
    if ( a[i] < a[i + 1] )
      Swap( a + i, a + i + 1 );
  bubbleSort( a, n - 1 );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
}

int main() {
  int a[] = { 3, 5, 4, 6, 7, 1, 2 };
  int size = sizeof a / sizeof *a;
  int i;

  printf( "Mang goc : " );
  for ( i = 0; i < size; ++i )
    printf( "%d ", a[i] );
  putchar( '\n' );

  bubbleSort( a, size );
  printf( "Mang giam: " );
  for ( i = 0; i < size; ++i )
    printf( "%d ", a[i] );
  putchar( '\n' );
  return 0;
}
```

Phân tích:

1. Sắp xếp tăng một mảng `A` theo thuật toán sắp xếp kiểu nổi bọt (bubble sort).
2. Kích thước đệ quy: số phần tử của mảng `A`.
3. Trường hợp cơ sở: khi mảng `A` có duy nhất một phần tử, xem như đã sắp xếp, dừng đệ quy.
4. Trường hợp tổng quát: dựa vào cấu trúc đệ quy của mảng, thao tác sắp xếp mảng `A` được mô tả đệ quy như sau:

`Sort(A, n)`	=	`Sort(A, n - 1)`	+	`A[n-1]`
Sắp xếp mảng `A`	=	Sắp xếp mảng con từ `A[0] -> A[n-2]`		Phần tử cuối `A[n-1]` đã sắp xếp

Thao tác sắp xếp giảm đúng *phần tử cuối* của một mảng (con) theo kiểu *nổi bọt* như sau: so sánh để hoán chuyển từng cặp phần tử kế tiếp nhau kể từ đầu mảng (con) đến cuối mảng (con); quá trình này làm phần tử nhỏ nhất của mảng (con) "nổi bọt" thành phần tử cuối mảng (con).

Bài 149: (trang 42)

```
#include <stdio.h>

int isAsc( int* a, int n ) {
  if ( n < 2 ) return 1;
  if ( a[n - 1] < a[n - 2] ) return 0;
  return isAsc( a, n - 1 );
}

int BSearch( int* a, int x, int left, int right ) {
  if ( left > right ) return -1;
  else {
    int m = ( left + right ) / 2;
    if ( a[m] == x ) return m;
    else
      if ( a[m] > x ) return BSearch( a, x, left, m - 1 );
      else            return BSearch( a, x, m + 1, right );
  }
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
int main() {
  int a[] = { 2, 3, 4, 5, 6, 7 };
  int size = sizeof a / sizeof *a;
  int i, x;

  for ( i = 0; i < size; ++i )
    printf( "%d ", a[i] );

  if ( isAsc( a, size ) ) {
    printf( "\nNhap x: " );
    scanf( "%d", &x );
    int pos = BSearch( a, x, 0, size - 1 );
    if ( pos != -1 )
      printf( "a[%d]\n", pos );
    else
      printf( "Khong tim thay\n" );
  }
  else printf( "Mang chua duoc sap xep\n" );
  return 0;
}
```

Giải thuật tìm kiếm nhị phân (Binary Search):

a) Được thực hiện trên mảng sắp xếp tăng `a`: phần tử đầu có chỉ số `left`, phần tử cuối có chỉ số `right`.

Nếu `left > right`, dừng tìm kiếm, trả về `-1` (không vị trí nào trong mảng). Đây là điểm dừng đệ quy.

b) Phần tử giữa mảng `a[m]` được so sánh với trị `x` cần tìm.

c) Nếu `x < a[m]` ta *tìm kiếm nhị phân* `x` *trong mảng con* bên trái: phần tử đầu có chỉ số `left`, phần tử cuối có chỉ số `m - 1`.

Nếu `x > a[m]` ta *tìm kiếm nhị phân* `x` *trong mảng con* bên phải: phần tử đầu có chỉ số `m + 1`, phần tử cuối có chỉ số `right`.

Hai phát biểu trên mang tính đệ quy.

Nếu `x = a[m]` ta đã tìm được `x`, trả về chỉ số của `x`. Dừng tìm kiếm. Đây là điểm dừng đệ quy.

Ví dụ: tìm `x` trong mảng sắp xếp tăng sau.

Phân tích:

1. Tìm kiếm `BSearch` một phần tử trong một mảng `a` sắp xếp tăng.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

2. Kích thước đệ quy: số phần tử của mảng a, biên trái left, biên phải right. m là chỉ số của phần tử giữa mảng:
m = (left + right) / 2;
3. Trường hợp cơ sở:
- Nếu left > right, trả về -1.
- Nếu x = a[m], đã tìm được x, trả về chỉ số của x.
4. Trường hợp tổng quát:
- Nếu x < a[m], tìm kiếm BSearch mảng con bên trái, biên trái left, biên phải m - 1.
- Nếu x > a[m], tìm kiếm BSearch mảng con bên trái, biên trái m + 1, biên phải right.
Cài đặt cho hàm BSearch() trên dùng nhiều if ... else để tường minh, có thể viết gọn hơn:

```
int BSearch( int* a, int x, int left, int right ) {
  int m = ( left + right ) / 2;
  if ( left > right ) return -1;
  if ( a[m] == x ) return m;
  return ( a[m] > x ) ? BSearch( a, x, left, m - 1 )
                      : BSearch( a, x, m + 1, right );
}
```

Hàm isAsc() thiết kế dựa trên phân tích sau:
1. Kiểm tra một mảng có phải là mảng sắp xếp tăng hay không.
2. Kích thước đệ quy: số phần tử của mảng.
3. Trường hợp cơ sở:
- Mảng chỉ có 1 phần tử (số phần tử < 2), xem như mảng tăng, trả về TRUE.
- Phát hiện hai phần tử kế tiếp nhau không đúng thứ tự tăng, trả về FALSE.
4. Trường hợp tổng quát: mô tả đệ quy như sau

isAsc(a, n)	=	isAsc(a, n - 1)	+	a[n-1]
Kiểm tra mảng a	=	Kiểm tra mảng con từ a[0] -> a[n-2]		Phần tử cuối a[n-1] đúng thứ tự (lớn hơn a[n-2])

Bài 150: (trang 43)

```
#include <stdio.h>

int OddSum( int* a, int n ) {
  if ( n < 1 ) return 0;
  return ( a[n - 1] % 2 ) ? a[n - 1] + OddSum( a, n - 1 )
                          : OddSum( a, n - 1 );
}

int main() {
  int a[] = { 2, 3, 4, 5, 6, 7 };
  int size = sizeof a / sizeof *a;
  int i;

  for ( i = 0; i < size; ++i )
    printf( "%d ", a[i] );
  printf( "\nTong cac phan tu le: %d\n", OddSum( a, size ) );
  return 0;
}
```

Phân tích:
1. Tính tổng các phần tử có trị lẻ trong một mảng.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

2. Kích thước đệ quy: số phần tử của mảng.
3. Trường hợp cơ sở: Mảng có `0` phần tử (số phần tử `< 1`), tổng các trị lẻ bằng `0`.
4. Trường hợp tổng quát:

- Phần tử cuối mảng có trị lẻ:

`OddSum(a, n)`	=	`a[n-1]`	+	`OddSum(a, n-1)`
Tổng trị lẻ mảng `a`	=	Trị của `a[n-1]` vì `a[n-1]` lẻ		Tổng trị lẻ của mảng con còn lại (từ `a[0]->a[n-2]`)

- Phần tử cuối mảng có trị chẵn:

`OddSum(a, n)`	=	`0`	+	`OddSum(a, n-1)`
Tổng trị lẻ mảng `a`	=	Không tính `a[n-1]` vì `a[n-1]` chẵn		Tổng trị lẻ của mảng con còn lại (từ `a[0]->a[n-2]`)

Các bài tập đệ quy trên mảng thường xét từ phần tử cuối ngược lên để bảo đảm `a[0]` luôn là phần tử đầu mảng con, thuận tiện và tự nhiên. Tuy vậy ta vẫn có thể xét các phần tử kể từ đầu mảng về sau nếu nắm vững thao tác con trỏ trên mảng.

Bài tập: dùng giải thuật đệ quy, in các phần tử lẻ trong mảng, theo thứ tự từ *đầu mảng* trở về sau.

```
void PrintOdd( int* a, int n ) {
  if ( n < 1 ) return;
  if ( a[0] % 2 ) {
    printf( "%d ", a[0] );
    PrintOdd( a + 1, n - 1 );
  }
  else PrintOdd( a + 1, n - 1 );
}
```

Giải pháp khác, dùng đệ quy đầu:

```
void PrintOdd( int* a, int n ) {
  if ( n < 1 ) return;
  if ( a[n - 1] % 2 ) {
    PrintOdd( a, n - 1 );
    printf( "%d ", a[n - 1] );
  }
  else PrintOdd( a, n - 1 );
}
```

Bài 151: (trang 43)

```
#include <stdio.h>

int Max( int* a, int n ) {
  if ( n < 2 ) return 0;
  return ( a[n - 1] > a[Max( a, n - 1 )] ) ? n - 1 : Max( a, n - 1 );
}

int main() {
  int a[] = { -2, 3, -4, -5, 6, -8 };
  int i;
  int size = sizeof a / sizeof *a;

  for ( i = 0; i < size; ++i )
    printf( "%d ", a[i] );

  i = Max( a, size );
  printf( "\nMax = a[%d] = %d\n", i, a[i] );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  return 0;
}
```

Phân tích:
1. Trả về chỉ số của phần tử có trị lớn nhất trong mảng không rỗng.
2. Kích thước đệ quy: số phần tử của mảng.
3. Trường hợp cơ sở: Mảng có `1` phần tử `a[0]` (số phần tử `< 2`), phần tử này có trị lớn nhất, trả về chỉ số `0`.
4. Trường hợp tổng quát: phát biểu đệ quy
Trị lớn nhất của mảng `a` (từ `a[0]` → `a[n-1]`), `Max(a, n)`, là số lớn hơn trong phép so sánh giữa phần tử cuối `a[n-1]` và *trị lớn nhất của mảng con* còn lại (từ `a[0]` → `a[n-2]`) tức trị `a[Max(a, n - 1)]`. Phép so sánh này trả về chỉ số của trị lớn nhất tương ứng.

Bài 152: (trang 43)

```
#include <stdio.h>

int isSym( int* a, int left, int right ) {
  if ( left > right ) return 1;
  if ( a[left] != a[right] ) return 0;
  return isSym( a, left + 1, right - 1 );
}

int main() {
  int a[] = { 7, -2, 3, 4, 3, -2, 7 };
  int i;
  int size = sizeof a / sizeof *a;

  for ( i = 0; i < size; ++i )
    printf( "%d ", a[i] );

  if ( isSym( a, 0, size - 1 ) ) printf( "\nMang doi xung\n" );
  else printf( "\nMang khong doi xung\n");
  return 0;
}
```

Phân tích:
1. Kiểm tra tính đối xứng của một mảng với `left` là biên trái, `right` là biên phải.
2. Kích thước đệ quy: số phần tử của mảng.
3. Trường hợp cơ sở:
- Phát hiện hai phần tử nằm tại các vị trí đối xứng `left` và `right` không cùng trị, trả về `FALSE`.
- `left > right`, mảng đã kiểm tra thành công, trả về `TRUE`.
4. Trường hợp tổng quát:
Kết quả *kiểm tra tính đối xứng của mảng* `a` (từ `left` → `right`) là kết quả kiểm tra `a[left] = a[right]` thành công và *kiểm tra tính đối xứng của mảng con* còn lại (từ `left+1` → `right-1`).
Ta cũng có thể thiết kế hàm `isSym()` một cách "quen thuộc", nghĩa là so sánh phần tử đầu với phần tử cuối, nhận tham số là mảng và số phần tử của mảng, … như sau:
`int isSym( int* a, int n );`
`a` là mảng cần kiểm tra, `n` là kích thước của mảng.

```
int isSym( int* a, int n ) {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  if ( n < 2 ) return 1;
  if ( a[0] != a[n - 1] ) return 0;
  return isSym( a + 1, n - 2 );
}
```

Mỗi lần đệ quy ta giảm n (kích thước đệ quy) đi 2 đơn vị do đã kiểm tra xong 2 phần tử của mảng.

Bài 153: (trang 43)

```
#include <stdio.h>

float NegAvg( int* a, int n ) {
  static int count = 0;
  static float sum = 0;
  if ( n < 1 ) return ( !count )? 0 : sum / count;
  if ( a[n - 1] < 0 ) {
    sum += a[n - 1];
    count++;
  }
  return NegAvg( a, n - 1 );
}

int main() {
  int a[] = { -2, 3, -4, -5, 6, -8 };
  int i;
  int size = sizeof a / sizeof *a;

  for ( i = 0; i < size; ++i )
    printf( "%d ", a[i] );

  printf( "\nTrung binh cong cac phan tu am: %g\n", NegAvg( a, size ) );
  return 0;
}
```

Có hai trị thay đổi liên tục mà hàm NegAvg() phải truyền từ khi bắt đầu đến kết thúc đệ quy là:
sum: tổng các số nguyên âm có trong mảng.
count: số các số nguyên âm có trong mảng.
Khi kết thúc đệ quy (trường hợp cơ sở, n < 1), hai trị này sẽ dùng tính tổng trung bình cộng các số nguyên âm.
Do yêu cầu của bài tập, hai trị luôn thay đổi này *không được truyền như tham số* giữa các lần gọi đệ quy. Vì vậy ta khai báo chúng như là biến static để những thay đổi được truyền cho các lần gọi sau.
Chú ý, do tính chất của biến static, hàm NegAvg() chỉ đúng với lần gọi đầu tiên từ hàm main(). Cách viết đệ quy sau ổn hơn:

```
float NegAvg( int* a, int n, int count, float sum ) {
  if ( n < 1 ) return ( !count )? 0 : sum / count;
  if ( a[n - 1] < 0 ) {
    sum += a[n - 1];
    count++;
  }
  return NegAvg( a, n - 1, count, sum );
}
/* gọi hàm */
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
printf( "\nTrung binh cong cac phan tu am: %g\n", NegAvg( a, size, 0, 0 ) );
```

Bài 154: (trang 43)

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

long fact( int n ) {
  if ( n == 0 ) return 1;
  return ( n * fact( n - 1 ) );
}

int strtoint( char *s ) {
  int n = 0;
  while ( isspace( *s ) ) s++;
  if ( !isdigit( *s ) ) return -1;
  while ( isdigit( *s ) ) n = n * 10 + *s++ - '0';
  return *s ? -1 : n;
}

int main( int argc, char *argv[] ) {
  int i, rv = 0;

  for ( i = 1; i < argc; ++i ) {
    int n  = strtoint( argv[i] );
    if ( n == -1 && ++rv ) printf( "%s: so khong hop le\n", argv[i] );
    else                   printf( "%d! = %ld\n", n, fact( n ) );
  }
  return rv;
}
```

Tham số dòng lệnh là các chuỗi nên trước khi tính giai thừa cần phải chuyển sang số. Hàm `strtoint()` đảm nhận việc này bằng cách dùng các hàm trong `ctype.h`, chuyển chuỗi tham số dòng lệnh thành một số *nguyên dương*. Hàm `strtoint()` trả về `-1` nếu phát hiện ký tự không hợp lệ. Mã lỗi của hàm `main()`, `rv`, trả về số tham số dòng lệnh nhập không hợp lệ.

Hàm `fact()` dùng giải thuật đệ quy để tính giai thừa, được viết dễ dàng dựa trên biểu thức truy hồi:

$$F_n \begin{cases} 1 & n = 0 \\ nF_{n-1} & n > 0 \end{cases}$$

Chú ý hàm `fact()` không tính được giai thừa số lớn, chỉ tính khoảng đến `32!`

Bài 155: (trang 43)

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

int Fi( int n ) {
  return ( n < 3 ) ? 1 : Fi( n - 1 ) + Fi( n - 2 );
}

int main() {
  double a, b;
  int n;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  do {
    printf( "Nhap n (0 < n < 40): " );
    scanf( "%d", &n );
  } while( n <= 0 || n >= 40 );

  printf( "De quy        : Fi(%d) = %d\n", n, Fi( n ) );

  a = ( 1 + sqrt( 5 ) ) / 2;
  b = 1 - a;
  printf( "Cong thuc dong: Fi(%d) = %.f\n", n,
           ( pow( a, n ) + pow( b, n ) ) / sqrt( 5 ) );
  return 0;
}
```

Hàm đệ quy `Fi()` được viết dễ dàng dựa trên biểu thức truy hồi:

$$F_n = \begin{cases} 1 & n = 1,2 \\ F_{n-1} + F_{n-2} & n > 2 \end{cases}$$

Tuy nhiên, do đệ quy nhị phân nên số lần gọi đệ quy rất lớn, chỉ dùng tính số Fibonacci với n nhỏ. Thường thay thế bằng thuật toán quy hoạch động.

Tham khảo thêm: biểu thức dạng đóng của số Fibonacci được tính như sau:

Biểu thức truy hồi của số Fibonacci:

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ F_n - F_{n-1} - F_{n-2} = 0 \quad n > 1 \end{cases}$$

Đặt $F_n = r^2$, từ $F_n - F_{n-1} - F_{n-2} = 0$ ta có phương trình đặc trưng:

$$r^2 - r - 1 = 0$$

Có 2 nghiệm đặc trưng: $r = \dfrac{1 \pm \sqrt{5}}{2}$ nên có lời giải tổng quát cho biểu thức đệ quy:

$$F_n = c_1\left(\frac{1+\sqrt{5}}{2}\right)^n + c_2\left(\frac{1-\sqrt{5}}{2}\right)^n$$

Xác định trị của các hằng số dựa vào các điều kiện đầu:

$$\begin{cases} F_0 = c_1 + c_2 \\ F_1 = c_1\left(\dfrac{1+\sqrt{5}}{2}\right) + c_2\left(\dfrac{1-\sqrt{5}}{2}\right) \end{cases} \Rightarrow \begin{cases} c_1 = \dfrac{1}{\sqrt{5}} \\ c_2 = -\dfrac{1}{\sqrt{5}} \end{cases}$$

Suy ra: $F_n = \dfrac{[(1+\sqrt{5})/2]^n - [(1-\sqrt{5})/2]^n}{\sqrt{5}}$

Đặt: $\varphi = \dfrac{1+\sqrt{5}}{2}$, $\psi = 1 - \varphi$, ta có biểu thức dạng đóng của số Fibonacci:

$$F_n = \frac{\varphi^n - \psi^n}{\sqrt{5}}$$

Bài 156: (trang 44)

```
#include <stdio.h>

double F( int n ) {
  if ( n == 103 ) return 103;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  return n + 1 / F( n + 2 );
}

int main() {
  printf( "F = %g", F( 1 ) );
  putchar( '\n' );
  return 0;
}
```

Xét phân số liên tục:

$$F = 1 + \cfrac{1}{3 + \cfrac{1}{5 + \cfrac{1}{\cdots \; 101 + \cfrac{1}{103}}}}$$

Dễ dàng nhận thấy công thức truy hồi của phân số liên tục trên:

$F_n = n + \dfrac{1}{F_{n+2}}, n \in [1, 101], n$ lẻ

Điều kiện đầu: $F_{103} = 103$

Đây là bài toán đệ quy kiểu chặn trên: bắt đầu với kích thước đệ quy $n = 1$, sau mỗi lần gọi đệ quy, n tăng thêm 2 đơn vị. Đến khi $n = 103$, bài toán tính được mà không cần đệ quy nhờ điều kiện đầu.

Bài 157: (trang 44)

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

int detMatrix( int* a, int n ) {
  int j, k, l;
  int sign = 1, sum = 0;
  if ( n == 1 ) return ( a[0] );
  for ( j = 0 ; j < n ; ++j, sign = -sign ) {
    /* tính A1j (chứa trong mảng a1) để truyền trong lời gọi đệ quy */
    int* a1 = ( int * )calloc( (n-1) * (n-1), sizeof( *a1 ) );
    for ( l = 0, k = n; k < n * n; ++k )
      if ( k % n != j ) a1[l++] = a[k];
    /* dùng công thức truy hồi để tính detA */
    sum += sign * a[j] * detMatrix( a1, n - 1 );
    free( a1 );
  }
  return sum;
}

int main( ) {
  int *a, n, i, j;

  printf( "Nhap bac ma tran: " );
  scanf( "%d", &n );

  a = ( int* )calloc( n * n, sizeof( *a ) );
  srand( time( NULL ) );
  for ( i = 0; i < n; ++i, putchar( '\n' ) )
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    for ( j = 0; j < n; ++j )
      printf( "%5d", a[i * n + j] = rand() % 21 - 10 );

  printf( "det(A) = %d\n", detMatrix( a, n ) );
  free( a );
  return 0;
}
```

Bài tập đã được cung cấp công thức truy hồi nên thuận lợi cho việc áp dụng giải thuật đệ quy:

$$\det(A) = \begin{cases} a_{11} & n = 1 \\ \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \det(A_{1j}) & n > 1 \end{cases}$$

Tuy nhiên vấn đề ở đây là việc thể hiện các ma trận `A` và A_{1j}. Ta gặp hàng loạt vấn đề: cấp phát động và giải phóng mảng 2 chiều (xem bài 99, trang 163), xóa dòng xóa cột để tạo ma trận mới (xem bài 94, trang 157), …

Để đơn giản, dùng mảng một chiều `a` (`n x n`) để chứa ma trận `A` và mảng một chiều `a1` (`(n - 1) x (n - 1)`) để chứa ma trận A_{1j}. Khi đó, trong công thức truy hồi:

a_{11} chính là `a[0]` của ma trận `a`.

a_{1j} chính là `a[j]` của ma trận `a`.

Xem hình dưới để hiểu cách tạo ma trận A_{1j} từ ma trận `A`.

Trong mảng một chiều `a` chứa ma trận `A`, ta duyệt các phần tử với `k` bắt đầu từ `n` (do đã xóa dòng đầu); sao chép những phần tử có chỉ số `k` với `k % n != j` vào mảng `a1` chứa ma trận. Biến `l` dùng làm biến đếm chỉ số cho mảng `a1`.

```
for ( l = 0, k = n; k < n * n; ++k )
  if ( k % n != j ) a1[l++] = a[k];
```

Công thức `k % n != j` nghĩa là "`k` giới hạn trong đoạn `[0, n)` và khác `j`".

Đến đây ta có thể viết hàm đệ quy:

- Trường hợp cơ bản: ma trận chỉ có một phần tử (a_{11}), trả về phần tử này, chính là `a[0]` của `a`.
- Trường hợp tổng quát: dùng công thức truy hồi:

$$\det(A) = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \det(A_{1j})$$

Trong vòng lặp theo `j`, cần xác định:

- Dấu của biểu thức: thay cho công thức phức tạp: $(-1)^{1+j}$ dùng biến `sign`, sẽ *đổi dấu* sau mỗi vòng lặp, khởi tạo dấu dương (do $(-1)^{1+j}$ bằng `1` với `j` ban đầu bằng `1`) .
- a_{1j} chính là `a[j]` của ma trận `a`.
- A_{1j} cho lời gọi đệ quy `det(A1j)`: tính như hướng dẫn trên.
- Lời gọi đệ quy với A_{1j}.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 158: (trang 44)

```
#include <stdio.h>

int gcd( int, int );        /* khai báo forward */

int gcdr( int m, int n ) {
  if ( !m ) return n;
  if ( !n ) return m;
  return gcd( n, m % n );
}

int gcd( int m, int n ) {
  if ( !m && !n ) return -1;
  if ( m < 0 ) m = -m;
  if ( n < 0 ) n = -n;
  return ( gcdr( m, n ) );
}

int main() {
  int m, n, g;

  printf( "Nhap hai so: " );
  scanf( "%d%d", &m, &n );

  printf( "GCD( %d, %d ) ", m, n );
  if ( ( g = gcd( m, n ) ) == -1 )
    printf( ": khong xac dinh\n" );
  else printf( "= %d\n", g );
  return 0;
}
```

USCLN (gcd - greatest common divisor) được tính bằng thuật toán Euclid, dùng công thức truy hồi:

$$\begin{cases} gcd(0, b) = b \\ gcd(a, b) = gcd(b \text{ MOD } a, a) \end{cases}$$

Hàm đệ quy `gcdr()` được xây dựng dựa trên công thức này.
Với trường hợp tham số truyền vào là số âm, dùng luật sau:

$gcd(a, b) = gcd(-a, b)$

Hàm không đệ quy `gcd()` được xem như một hàm bọc (wrapper) hàm `gcdr()` để lọc trường hợp tham số truyền vào là số âm, trước khi chuyển nó cho hàm `gcdr()`.
Nếu ta dùng công thức truy hồi sau:

$$\begin{cases} gcd(0, b) = b \\ gcd(a, b) = gcd(b, a - b) \end{cases}$$

Trị `a - b` có thể `< 0`, nghĩa là phải giải quyết bằng luật: $gcd(a, b) = gcd(-a, b)$
Khi đó ta có thể tạo hai hàm `gcdr()` và `gcd()` như là một cặp đệ quy tương hỗ, phối hợp gọi lẫn nhau để giải quyết vấn đề.
Xem thêm một ví dụ về đệ quy tương hỗ trong bài 161 (trang 237).
Khai báo forward giúp trình biên dịch "hình dung" được trị trả về và danh sách tham số của hàm mặc dù trình biên dịch chưa thấy định nghĩa của hàm.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 159: (trang 45)

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

void shrink( char* buffer ) {
  char *p, *n;
  for ( n = p = buffer; *p; ++p )
    if ( isalpha( *p ) )
      *n++ = tolower( *p );
  *n = '\0';
}

int ispal( char* first, char* last ) {
  if ( first >= last ) return 1;
  if ( *first != *last ) return 0;
  return ispal( first + 1, last - 1 );
}

int main() {
  char buffer [1000];
  while ( fgets( buffer, sizeof( buffer ), stdin ) != NULL ) {
    shrink( buffer );
    size_t len = strlen( buffer );
    if ( len == 0 )
      printf( "-> Khong nhan duoc chuoi\n" );
    else
      if ( ispal( buffer, &buffer[len - 1] ) )
        printf( "-> Palindrome\n" );
      else printf( "-> Khong phai palindrome\n" );
  }
  return 0;
}
```

Để tránh những rắc rối khi kiểm tra tính đối xứng của chuỗi như: các ký tự khoảng trắng, ký tự không phải alphabet, ký tự chữ hoa, … ta dùng hàm `shrink()` để "chuẩn hóa" chuỗi trước. Hàm này sẽ loại bỏ các ký tự khác alphabet (dùng kỹ thuật xóa mảng trình bày trong bài 64, trang 119), các ký tự còn lại đều chuyển thành chữ thường. Kết quả lưu trong mảng `buffer`.
Sau đó việc kiểm tra tính đối xứng (hàm `ispal()`) thực hiện bình thường như kiểm tra tính đối xứng của một mảng ký tự bằng đệ quy, xem bài 152 (trang 229).

Bài 160: (trang 45)

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

#define Swap( a, b ) { char c = a; a = b; b = c; }

void permutation( char* s, int l, int r ) {
  if ( l == r ) printf( "%s ", s );
  else {
    int i;
    for ( i = l; i <= r; ++i ) {
      Swap( s[i], s[l] );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
      permutation( s, l + 1, r );
      Swap( s[i], s[l] );
    }
  }
}

int main() {
  int n, i;
  char* s;

  printf( "Nhap n: " );
  scanf( "%d", &n );
  s = ( char* )calloc( n + 1, 1 );
  for ( i = 0; i < n; ++i )
    s[i] = i + 'A';
  permutation( s, 0, n - 1 );
  return 0;
}
```

Xét dãy ký tự của chuỗi từ `s[0]` đến `s[n - 1]` (`n` ký tự, `n = strlen(s)`).
Ta tìm cách tạo ra "bộ" hoán vị *bắt đầu* bằng `s[0]`, sau đó tạo ra "bộ" hoán vị *bắt đầu* bằng `s[1]`, …
"Bộ" những hoán vị bắt đầu bằng `s[0]` đều có ký tự đầu là `s[0]`, phần đuôi là hoán vị của `n - 1` phần tử còn lại (từ `s[1]` đến `s[n - 1]`). Như vậy ta đã có được lập luận truy hồi để đưa vấn đề từ kích thước `n` (ký tự) về kích thước nhỏ hơn `n - 1` (ký tự).
Hàm `permutation(s, l, r)` tạo các "bộ" hoán vị với ký tự bắt đầu là `s[i]`: `s[l]`, `s[l + 1]`, …, `s[r]` như sau:
- Hoán chuyển ký tự `s[i]` với `s[l]` nhằm tạo "bộ" hoán vị với ký tự đầu `s[i]`.
- Các hoán vị trong "bộ" hoán vị này đều có ký tự đầu là `s[i]`, phần đuôi là hoán vị của các ký tự còn lại, tạo bằng cách gọi đệ quy với quy mô nhỏ hơn: `permutation(s, l + 1, r)`.
- Sau khi gọi đệ quy ở bước 2, chuỗi `s` vẫn giữ nguyên như trước khi gọi đệ quy. Ta cần thực hiện hoán chuyển ký tự *ngược với bước 1* để trả chuỗi `s` về như ban đầu, chuẩn bị tạo "bộ" hoán vị kế tiếp.
Bây giờ ta xét trường hợp cơ sở để dừng đệ quy: nếu `l = r` thì đoạn cần hoán vị chỉ có 1 ký tự, chỉ cần in ra.

Bài 161: (trang 45)

```
#include <stdio.h>
#include <math.h>
#define eps 5e - 4

double C( double );          /* khai báo forward */

double S( double x ) {
  if ( fabs( x ) < eps ) return x * ( 1 - x * x / 6 );
  return ( 4 * C( x / 3 ) * C( x / 3 ) - 1 ) * S( x / 3 );
}

double C( double x ) {
  if ( fabs( x ) < eps ) return ( 1 - x * x / 2 );
  return ( 1 - 4 * S( x / 3 ) * S( x / 3 ) ) * C( x / 3 );
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
int main() {
  double x = -2.1;

  printf( "Doi chung tinh bang math.h trong ()\n" );
  printf( "sin(%g) = %14.10g ( %.10g )\n", x, S( x ), sin( x ) );
  printf( "cos(%g) = %14.10g ( %.10g )\n", x, C( x ), cos( x ) );
  return 0;
}
```

Bài tập này dùng đệ quy tương hỗ: hai (hay nhiều) hàm *gọi nhau một cách đệ quy*. Tính tương hỗ được thể hiện khi hàm `f1` gọi hàm `f2` và hàm `f2` gọi lại hàm `f1` với *quy mô nhỏ hơn*, ta gọi là "vào lại" (re-entrant).

Cũng giống như đệ quy thông thường, cách đệ quy này yêu cầu mỗi hàm gọi có trường hợp cơ sở (có điểm dừng) và kích thước bài toán thay đổi (thường là giảm) sau mỗi lời gọi đệ quy.

Phân tích:

1. Tính `sin( x )` và `cos( x )` dựa vào đệ quy tương hỗ.
2. Kích thước đệ quy: trị `x`, sẽ giảm 1/3 sau mỗi lần gọi đệ quy. Nói cách khác, hai hàm này gọi qua lại với nhau, mỗi lần gọi kích thước đệ quy giảm dần, tiến đến trường hợp cơ sở.
3. Trường hợp cơ sở: với `x` rất nhỏ ($< 5.10^{-4}$) ta không dùng cặp đồng nhất thức đệ quy tương hỗ để tính mà dùng cặp đồng nhất thức xấp xỉ *không đệ quy* để tính:

$$\begin{cases} \sin x \approx x - x^3 / 6 \\ \cos x \approx 1 - x^2 / 2 \end{cases}$$

và như vậy ta dừng luôn đệ quy tại đây.

4. Trường hợp tổng quát: với `x` lớn ta dùng cặp đồng nhất thức *đệ quy* tương hỗ:

$$\begin{cases} \sin 3x = (4\cos^2 x - 1)\sin x \\ \cos 3x = (1 - 4\sin^2 x)\cos x \end{cases}$$

Chú ý kích thước đệ quy giảm cho mỗi lời gọi đệ quy.

Bài 162: (trang 45)

```
#include <stdio.h>

void hanoi( int n, int source, int target ) {
  if ( n == 1 )
    printf( "Disk %d: [%d] -> [%d]\n", n, source, target );
  else {
    int between = 6 - ( source + target );
    hanoi( n - 1, source, between );
    printf( "Disk %d: [%d] -> [%d]\n", n, source, target );
    hanoi( n - 1, between, target );
  }
}

int main() {
  hanoi( 4, 1, 3 );
  return 0;
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Ta gọi `hanoi(n, source, target)` là phép chuyển `n` đĩa từ cọc `source` sang cọc `target`. Giả sử ta đã chuyển được `n - 1` đĩa (trừ đĩa lớn nhất) từ cọc `source` sang cọc trung gian `between` bằng phép chuyển `hanoi(n - 1, source, between)`, thì ta có thể dễ dàng thực hiện được `hanoi(n, source, target)` bằng ba thao tác:

- `hanoi(n - 1, source, between)`: chuyển `n - 1` đĩa (trừ đĩa lớn nhất) từ cọc `source` sang cọc trung gian `between`.
- Chuyển đĩa thứ `n` từ cọc `source` sang cọc `target`.
- `hanoi(n - 1, between, target)`: chuyển `n - 1` đĩa từ cọc trung gian `between` sang cọc `target`, chồng lên đĩa lớn nhất vừa chuyển sang.

Như vậy ta đã xác định được lập luận truy hồi cho trò chơi trên.
Đặt trị cho từng cọc: `source`(1), `between`(2), `target`(3). Ta có:
`source + between + target = 6`
Từ đó dễ dàng xác định được cọc `between` trong các lời gọi đệ quy khác (với `source` và `target` khác): `between = 6 - ( source + target )`
Bây giờ cần xác định trường hợp cơ sở (điều kiện đầu) để dừng đệ quy: nếu chỉ có một đĩa, thì đơn giản chuyển đĩa này trực tiếp từ cọc `source` sang cọc `target`.
Tham khảo thêm: giả sử số bước để chuyển `n` đĩa là `F(n)`.
Từ điều kiện đầu, ta có: `F(1) = 1`
Từ lập luận truy hồi trên, ta có: `F(n) = F(n-1) + 1 + F(n-1), n > 1`
Suy ra:
`F(1) + 1 = 2`
`F(n) + 1 = 2F(n-1) + 1 + 1 = 2(F(n-1) + 1), n > 1`
Vậy hàm `G(n) = F(n) + 1` chính là hàm mũ: $G(n) = 2^n$
$F(n) = 2^n - 1, n > 0$
Để chuyển `n` đĩa, cần $2^n - 1$ bước chuyển. Vì vậy không nên thực hiện bài tập với số lớn. Ví dụ để chuyển 64 đĩa, mỗi bước chuyển mất 1 giây, chúng ta cần 585 tỷ năm (vũ trụ hình thành khoảng 13,7 tỷ năm).
Ngoài giải thuật đệ quy, người ta còn giải bài tập này bằng nhiều giải thuật khác như giải thuật Binary, giải thuật mã Gray...

Bài 163: (trang 46)

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

double mypow( double x, int n ) {
  double xlast;
  if ( n < 0 ) return 1 / mypow( x, -n );
  if ( n == 0 ) return 1;
  if ( n == 1 ) return x;
  if ( n % 2 ) {
    xlast = mypow( x, ( n - 1 ) / 2 );
    return xlast * xlast * x;
  } else {
    xlast = mypow( x, n / 2 );
    return xlast * xlast;
  }
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
}

int main() {
  double x;
  int n;

  printf( "Nhap x, n: " );
  scanf( "%lf%d", &x, &n );

  printf( "mypow() : %lf\n", mypow( x, n ) );
  printf( "pow()   : %lf\n", pow( x, n ) );
  return 0;
}
```

Phân tích:

1. Tính lũy thừa x^n. Để tổng quát hơn, ta dùng biểu thức truy hồi:

$$x^n = \begin{cases} 1 / x^n & n < 0 \\ 1 & n = 0 \\ x & n = 1 \\ (x^k)(x^k) & n = 2k \quad k \in N \\ x(x^k)(x^k) & n = 2k + 1 \end{cases}$$

Trường hợp `n < 0` thường bị bỏ qua khi giải bài tập.

2. Kích thước đệ quy: `n`, sẽ giảm dần mỗi lần gọi đệ quy (trừ trường hợp `n < 0`).

3. Trường hợp cơ sở:

`n = 0`, x^n `= 1.`

`n = 1`, x^n `= x.` Trường hợp này chỉ dùng để giảm 1 lần đệ quy khi `n` lẻ, không cần thiết trong giải thuật.

4. Trường hợp tổng quát:

Dùng biểu thức truy hồi mô tả ở trên.

Bài 164: (trang 46)

```
#include <stdio.h>

typedef struct {
  double real, imag;
} Complex;

Complex add( Complex a, Complex  b ) {
  Complex t;
  t.real = a.real + b.real;
  t.imag = a.imag + b.imag;
  return t;
}

Complex sub( Complex a, Complex  b ) {
  Complex t;
  t.real = a.real - b.real;
  t.imag = a.imag - b.imag;
  return t;
}

Complex mul( Complex a, Complex  b ) {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  Complex t;
  t.real = a.real * b.real - a.imag * b.imag;
  t.imag = a.real * b.imag + a.imag * b.real;
  return t;
}

Complex div( Complex a, Complex  b ) {
  Complex t;
  double k = b.real * b.real + b.imag * b.imag;
  t.real = ( a.real * b.real + a.imag * b.imag ) / k;
  t.imag = ( a.imag * b.real - a.real * b.imag ) / k;
  return t;
}

Complex input() {
  Complex t;
  printf( "Nhap mot so phuc:\n" );
  printf( "  Phan thuc: " );
  scanf( "%lf", &t.real );
  printf( "  Phan ao  : " );
  scanf( "%lf", &t.imag );
  return t;
}

void output( Complex a ) {
  printf( "%.1lf%+.1lfi\n", a.real, a.imag );
}

int main() {
  Complex a, b;

  a = input();
  b = input();
  printf( "a + b = " );  output( add( a, b ) );
  printf( "a - b = " );  output( sub( a, b ) );
  printf( "a * b = " );  output( mul( a, b ) );
  printf( "a / b = " );  output( div( a, b ) );
  return 0;
}
```

Thông thường một structure được khai báo như sau:

`struct sComplex { double real, imag; };`

ta đã khai báo một kiểu dữ liệu mới: `struct sComplex`. Từ khóa `struct` bắt buộc kèm theo ở bất cứ nơi nào có dùng kiểu vừa khai báo (C++ bỏ qua quy tắc này). Để có tên kiểu mới "một từ", ta có thể dùng `typedef`:

`typedef struct sComplex Complex;`

bây giờ có thể dùng tên kiểu "một từ" `Complex` thay cho `struct sComplex`.

Từ kiến thức trên, các bài tập về structure trong tập sách này dùng một cách khai báo khá tiện dụng:

`typedef struct { double real, imag; } Complex;`

nghĩa là khai báo một structure *vô danh*, sau đó dùng `typedef` gán cho structure vô danh này một tên kiểu "một từ" `Complex`.

Thao tác chủ yếu trên dữ liệu kiểu structure là truy xuất các thành viên của structure thông qua toán tử `"."` hoặc `"->"`. Bạn cần chú ý thao tác truy xuất thành viên này.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Kiểu dữ liệu structure sẽ được mở rộng trong C++ thành khái niệm `class` và thao tác tốt trên structure sẽ giúp bạn làm việc tốt với `class` sau này.

Bài 165: (trang 46)

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

typedef struct {
  int num, denom;
} Fraction;

int gcd( int a, int b ) {
  return ( !a ) ? b : gcd( b % a, a );
}

Fraction reduce( Fraction a ) {
  Fraction t;
  int r = gcd( abs( a.num ), abs( a.denom ) );
  t.num = a.num / r;
  t.denom = a.denom / r;
  if ( t.denom < 0 ) {
    t.num = -t.num;
    t.denom = -t.denom;
  }
  return t;
}

Fraction add( Fraction a, Fraction b ) {
  Fraction t;
  t.num = a.num * b.denom + b.num * a.denom;
  t.denom = a.denom * b.denom;
  return reduce( t );
}

Fraction sub( Fraction a, Fraction b ) {
  Fraction t;
  t.num = a.num * b.denom - b.num * a.denom;
  t.denom = a.denom * b.denom;
  return reduce( t );
}

Fraction mul( Fraction a, Fraction b ) {
  Fraction t;
  t.num = a.num * b.num;
  t.denom = a.denom * b.denom;
  return reduce( t );
}

Fraction div( Fraction a, Fraction b ) {
  Fraction t;
  t.num = a.num * b.denom;
  t.denom = a.denom * b.num;
  return reduce( t );
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
void output( Fraction a ) {
  if ( a.denom == 1 ) printf( "%d\n", a.num );
  else                printf( "%d/%d\n", a.num, a.denom );
}

Fraction input() {
  Fraction t;
  do {
    printf( "Nhap tu so va mau so: " );
    scanf( "%d%d", &t.num, &t.denom );
  } while ( !t.denom );
  return reduce( t );
}

int main() {
  Fraction a, b;

  a = input();  b = input();
  printf( "a + b = " );  output( add( a, b ) );
  printf( "a - b = " );  output( sub( a, b ) );
  printf( "a * b = " );  output( mul( a, b ) );
  printf( "a / b = " );  output( div( a, b ) );
  return 0;
}
```

Ngoài các hàm chính, ta cần viết thêm hai hàm bổ trợ (helper): hàm `reduce()` gọi hàm `gcd()`:

- Hàm `gcd( a, b )` dùng lấy ước số chung lớn nhất của hai số `a` và `b`. Một vài cách đã trình bày trong bài 25 (trang 86) . Cách giải quyết trong bài tập này vẫn theo thuật toán Euclid:

$$\begin{cases} \text{gcd}(0, b) = b \\ \text{gcd}(a, b) = \text{gcd}(b \text{ MOD } a, a) \quad (a > 0) \end{cases}$$

- Hàm `reduce()` dùng tối giản một phân số, được thực hiện tương tự bài 26 (trang 88). Hàm này được gọi vào bất kỳ thời điểm nào có một phân số mới được tạo ra.

Bài 166: (trang 47)

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

typedef struct {
  double xc, yc;
  double R;
} CIRCLE;

int main() {
  CIRCLE a, b;
  double d;

  printf( "Nhap xc, yc va R cua C1: " );
  scanf( "%lf%lf%lf", &a.xc, &a.yc, &a.R );
  printf( "Nhap xc, yc va R cua C2: " );
  scanf( "%lf%lf%lf", &b.xc, &b.yc, &b.R );

  d = sqrt( ( b.xc - a.xc ) * ( b.xc - a.xc ) +
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
            ( b.yc - a.yc ) * ( b.yc - a.yc ) );

  if ( d == 0 ) {
    if ( a.R == b.R ) printf( "Trung nhau\n" );
    else if ( a.R < b.R ) printf( "C1 trong C2\n" );
         else printf( "C2 trong C1\n" );
  }
  else if ( d < a.R + b.R ) {
    if ( d == fabs( a.R - b.R ) ) printf( "Tiep xuc trong\n" );
    else if ( d < fabs( a.R - b.R ) ) printf( "Trong nhau\n" );
         else printf( "Cat nhau\n" );
  }
  else if ( d == a.R + b.R ) printf( "Tiep xuc ngoai\n" );
       else printf( "Ngoai nhau\n" );
  return 0;
}
```

Khi biện luận phức tạp nhiều trường hợp, nên vẽ lưu đồ để xác định đủ các trường hợp và viết code thuận tiện theo lưu đồ:

Bài 167: (trang 47)

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

typedef struct {
  double coef;
  int power;
} MONO;

int inputPoly( MONO** poly ) {
  int n, i;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  printf( "Nhap bac da thuc: " );
  scanf( "%d", &n );
  *poly = ( MONO* )calloc( n + 1, sizeof( MONO ) );
  if ( !*poly ) return -1;

  printf( "Nhap %d he so: ", n + 1 );
  for ( i = n; i >= 0; --i ) {
    scanf( "%lf", &(*poly)[i].coef );   /* nhập hệ số */
    (*poly)[i].power = i;               /* tự gán lũy thừa */
  }
  while ( (*poly)[n].coef == 0 ) n--;   /* đa thức suy biến */
  return n;
}

void outputPoly( MONO* poly, int n, char* s ) {
  int i;
  printf( "%s(x) = ", s );
  if ( poly[n].coef  && n > 1 )
    printf( "%gx^%d", poly[n].coef, poly[n].power );
  for ( i = n - 1; i > 1; --i )
    if ( poly[i].coef )
      printf( "%+gx^%d", poly[i].coef, poly[i].power );
  if ( poly[1].coef ) printf( "%+gx", poly[1].coef );
  if ( poly[0].coef ) printf( "%+g", poly[0].coef );
  putchar( '\n' );
}

MONO* mulPoly( MONO* poly1, MONO* poly2, int n1, int n2 ) {
  MONO* poly;
  int i, j;
  poly = ( MONO* )calloc( n1 + n2 + 1, sizeof( MONO ) );
  if ( !poly ) return NULL;

  for ( i = n1 + n2; i >= 0; --i ) {
    poly[i].coef = 0.0;
    poly[i].power = i;
  }

  for ( i = n1; i >= 0; --i )
    for ( j = n2; j >= 0; --j )
      poly[i + j].coef += poly1[i].coef * poly2[j].coef;
  return poly;
}

double valuePoly( MONO* poly, int n, double x ) {
  int i;
  double v;
  v = poly[n].coef;
  for ( i = n - 1; i >= 0; --i )
    v = v * x + poly[i].coef;
  return v;
}

int main() {
  MONO *poly1, *poly2, *poly;
  int n1, n2;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  double x;

  n1 = inputPoly( &poly1 );
  n2 = inputPoly( &poly2 );
  if ( n1 == -1 || n2 == -1 )
    { printf( "Loi cap phat\n" ); return 1; }
  outputPoly( poly1, n1, "P1" );
  outputPoly( poly2, n2, "P2" );

  poly = mulPoly( poly1, poly2, n1, n2 );
  if ( !poly )
    { printf( "Loi cap phat\n" ); return 1; }
  printf( "Da thuc ket qua:\n" );
  outputPoly( poly, n1 + n2, "P" );

  printf( "Nhap x: " );
  scanf( "%lf", &x );
  printf( "P(%g) = %g\n", x, valuePoly( poly, n1 + n2, x ) );
  free( poly1 );
  free( poly2 );
  free( poly );
  return 0;
}
```

Các hàm trong bài giải:

- Hàm `inputPoly()`: nhận một con trỏ (kiểu `MONO*`), cấp phát cho con trỏ này, trả về số phần tử của mảng quản lý bởi con trỏ vừa cấp phát hoặc trả về `-1` nếu cấp phát không thành công. Con trỏ `poly`, kiểu `MONO*`, truyền đến hàm thay đổi sau khi gọi hàm, nên được truyền *bằng con trỏ* đến hàm (xem bài 77, trang 135): `MONO** poly`, chú ý trong hàm `poly` luôn có dấu `*` kèm theo.

Sau khi nhập các hệ số và tự gán các chỉ số, cần kiểm tra hệ số của bậc lớn nhất để tránh trường hợp nhập hệ số `0` làm đa thức suy biến.

- Hàm `outputPoly()`: xuất đa thức ra màn hình. Để đa thức in ra có dạng tự nhiên, ta phân biệt các trường hợp: in đơn thức bậc lớn nhất - in đơn thức các bậc lớn hơn 1 - in đơn thức bậc 1 và bậc 0.

- Hàm `mulPoly()`: nhân hai đa thức được truyền đến và trả về đa thức kết quả. Cách nhân đa thức theo đúng hướng dẫn: tích của 2 đa thức 1 tham số `x`, `f(x)` bậc `m` và `g(x)` bậc `n` là: $f(x).g(x) = c_{m+n}x^{m+n} + c_{m+n-1}x^{m+n-1} + \ldots + c_1x + c_0$

Trong đó c_k bằng tổng các tích a_ib_j mà $i + j = k$ $(k = 0,1,\ldots,m + n)$

- Hàm `valuePoly()`: tính trị của đa thức bằng phương pháp Horner:

$$P(x) = ((\ldots((a_nx + a_{n-1})x + a_{n-2})x + \ldots + a_2)x + a_1)x + a_0$$

Có thể tính bằng vòng lặp như bài giải, hoặc tính bằng giải thuật đệ quy:

```
double valuePoly( MONO* poly, int n, int c, double x ) {
  if ( c == n ) return poly[n].coef;
  return valuePoly( poly, n, c + 1, x ) * x + poly[c].coef;
}
/* gọi hàm */
printf( "P(%g) = %g\n", x, valuePoly( poly, n1 + n2, 0, x ) );
```

Bài 168: (trang 48)

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

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
#include <stdlib.h>
#include <string.h>
#include <time.h>

typedef struct {
  char bookTitle[80];
  char isbn[15];
  char author[80];
  char publisher[32];
  struct tm* dateAdded;
} BOOK;

BOOK* inputBook() {
  BOOK* b;
  time_t now;
  printf( "Nhap thong tin sach:\n" );
  b = ( BOOK* )calloc( 1, sizeof( BOOK ) );
  if ( !b ) return NULL;

  printf( " Tua     > " );  fgets( b->bookTitle, 80, stdin );
  printf( " ISBN    > " );  fgets( b->isbn, 15, stdin );
  printf( " Tac gia > " );  fgets( b->author, 80, stdin );
  printf( " NXB     > " );  fgets( b->publisher, 32, stdin );
  time( &now );
  b->dateAdded = localtime( &now );
  return b;
}

void outputBook( BOOK* b ) {
  printf( "%s", b->bookTitle );
  printf( " %s %s", b->author, b->publisher );
  printf( " [upate: %02d-%02d-%d]\n", b->dateAdded->tm_mday,
    b->dateAdded->tm_mon + 1, b->dateAdded->tm_year + 1900 );
}

BOOK* findBook( BOOK* books[], int n, char* ISBN ) {
  int i;
  for ( i = 0; i < n; ++i )
    if ( strcmp ( books[i]->isbn, ISBN ) == 0 )
      return books[i];
  return NULL;
}

int main() {
  BOOK* books[1000] = { 0 };
  BOOK* t;
  char s[15];
  int n = 0;
  char c;
  do {
    c = 0;
    if ( ( books[n] = inputBook() ) != NULL ) n++;
    do {
      printf( "Tiep ( y/n )? " );
      scanf( "%1[yn]c", &c );
      while ( getchar() != '\n' ) { }
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    } while ( !c );
  } while ( c != 'n' );
  printf( "ISBN ? " );
  fgets( s, 15, stdin );
  printf( "Ket qua tim:\n" );
  if ( ( t = findBook( books, n, s ) ) != NULL )
    outputBook( t );
  else printf( "Khong tim thay\n" );
  n = 0;
  while ( books[n] ) free( books[n++] );
  return 0;
}
```

Để quản lý được nhiều sách, ta không dùng mảng các structure `BOOK` mà dùng mảng `books` chứa các con trỏ chỉ đến structure `BOOK`.
Khi nhập thông tin một quyển sách mới cần quản lý bằng hàm `inputBook()`, hàm này sẽ cấp phát và trả về một con trỏ chỉ đến structure chứa thông tin quyển sách mới đó. Con trỏ trả về sẽ được đặt vào mảng `books`. Truy xuất các thành viên của structure thông qua *con trỏ chỉ đến structure* đó được thực hiện bằng cách dùng toán tử `"->"`. Các thông tin dạng chuỗi đều nhận bằng `fgets()`, không cần loại bỏ ký tự `'\n'` vì chuỗi tìm kiếm cũng nhận bằng `fgets()` nên cũng chứa ký tự này.
Cấu trúc `tm`, kiểu dữ liệu `time_t` và cách dùng, xem bài tập 190 (trang 278).

Bài 169: (trang 48)

```
#include <stdio.h>
#include <string.h>
#define PAY1 15000
#define PAY2 10000

typedef struct {
  int h;
  int m;
} TIME;

typedef struct {
  char workerID[8];
  TIME t_in, t_out;
} timeCard;

long toMinute( TIME t ) {
  return t.h * 60 + t.m;
}

double pay( TIME a, TIME b ) {
  if ( a.h >= 1 && a.h <= 6 ) {
    if ( b.h < a.h ) return -1;
    if ( b.h <= 6 )              /* trường hợp 1 */
      return PAY1 * ( toMinute( b ) - toMinute( a ) ) / 60.0;
    if ( b.h <= 18 )             /* trường hợp 2 */
      return PAY1 * ( 6 * 60 - toMinute( a ) ) / 60.0 +
             PAY2 * ( toMinute( b ) - 6 * 60 ) / 60.0;
    return -1;
  }
  if ( a.h >= 6 && a.h <= 18 ) {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    if ( b.h < a.h ) return -1;
    if ( b.h <= 18 )            /* trường hợp 3 */
      return PAY2 * ( toMinute( b ) - toMinute( a ) ) / 60.0;
    if ( b.h <= 24 )            /* trường hợp 4 */
      return PAY2 * ( 18 * 60 - toMinute( a ) ) / 60.0 +
             PAY1 * ( toMinute( b ) - 18 * 60 ) / 60.0;
    return -1;
  }
  if ( a.h >= 18 && a.h <= 24 ) {
    if ( b.h >=1 && b.h <= 6 )  /* trường hợp 6 */
      return PAY1 * ( 24 * 60 - toMinute( a ) ) / 60.0 +
             PAY1 * ( toMinute( b ) - 1 * 60 ) / 60.0;
    if ( b.h <= 24 ) {          /* trường hợp 5 */
      if ( b.h < a.h ) return -1;
      return PAY1 * ( toMinute( b ) - toMinute( a ) ) / 60.0;
    }
    return -1;
  }
  return -1;
}

int main() {
  timeCard group[6];
  int i, n = 6;

  for ( i = 0; i < n; ++i ) {
    printf( "Nhap ID cong nhan %d: ", i + 1 );
    fgets( group[i].workerID, 8, stdin );
    char* t = strrchr(group[i].workerID, '\n' );
    if ( t != NULL ) *t = '\0';
    printf( "Vao (gio phut): " );
    scanf( "%d:%d", &group[i].t_in.h, &group[i].t_in.m );
    printf( "Ra (gio phut) : " );
    scanf( "%d:%d", &group[i].t_out.h, &group[i].t_out.m );
    while ( getchar() != '\n' ) { }
  }

  for ( i = 0; i < n; ++i ) {
    printf( "%8s  %02d:%02d  %02d:%02d   ",
            group[i].workerID,
            group[i].t_in.h, group[i].t_in.m,
            group[i].t_out.h, group[i].t_out.m );
    double d = pay( group[i].t_in, group[i].t_out );
    if ( d != -1 ) printf( "%-.lf\n", d );
    else printf( "%-s\n", "Nhap sai!" );
  }
  return 0;
}
```

Việc tính lương công nhân chỉ làm một ca (trường hợp 1, 3, 5 trong hình dưới) không khó, chỉ cần tính thời gian chênh lệch giữa hai thời điểm ra vào ca (quy thành bằng phút để tính, xem bài 18 (trang 81), chuyển thành giờ rồi nhân với lương quy định. Nếu công nhân làm bắc qua hai ca, tính lương trở nên phức tạp hơn nhiều. Tuy nhiên do có quy định không làm vượt quá hai ca nên chỉ có 6 trường hợp cần tính lương được mô tả trong hình dưới đây:

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

6 trường hợp này chia làm 3 nhóm chính dựa vào *thời điểm vào ca*: 1-2, 3-4, 5-6. Các trường hợp 1, 2, 3, 4 và 5 cần chú ý thời điểm ra ca không được nhỏ hơn thời điểm vào ca, điều kiện này như là một "chặn dưới" cho các điều kiện về thời điểm ra ca. Riêng trường hợp 6 cần biện luận điều kiện chính xác để tránh xung đột điều kiện với trường hợp 5.

Bài 170: (trang 49)

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

typedef struct {
  unsigned face: 4;
  unsigned suit: 2;
} Card;

char* Face[] = { "Ach", "Hai", "Ba", "Bon", "Nam", "Sau",
                 "Bay", "Tam", "Chin", "Muoi", "Boi", "Dam", "Gia" };
char* Suit[] = { "Co", "Ro", "Chuon", "Bich" };

void fill( Card* Deck ) {
  int i;
  for ( i = 0; i < 52; ++i ) {
    Deck[i].face = i % 13;
    Deck[i].suit = i / 13;
  }
}

void shuffle( Card* Deck ) {
  srand( time( NULL ) );
  for ( int i = 0; i < 52; ++i ) {
    int j = rand() % 52;
    Card t = Deck[i]; Deck[i] = Deck[j]; Deck[j] = t;
  }
}

void deal( Card* Deck ) {
  int i;
  for ( i = 0; i < 52; i += 4 )
    printf( "%5s %-6s%5s %-6s%5s %-6s%5s %-5s\n",
            Face[Deck[i].face], Suit[Deck[i].suit],
            Face[Deck[i+1].face], Suit[Deck[i+1].suit],
            Face[Deck[i+2].face], Suit[Deck[i+2].suit],
            Face[Deck[i+3].face], Suit[Deck[i+3].suit] );
}

int main() {
  Card deck[ 52 ];

  fill( deck );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  shuffle( deck );
  deal( deck );
  return 0;
}
```

Thông tin về một lá bài gồm nước (`Face`) và chất (`Suit`) đi đôi với nhau nên ta dùng structure `Card` để lưu là hợp lý. Mảng `deck` các structure `Card` dùng lưu thông tin một bộ bài. Để tiết kiệm không gian lưu các thành viên của structure `Card` được lưu dưới dạng số trong các `bit field` (xem bài 171, trang 251); khi xuất lá bài nước và chất cụ thể của lá bài được tham chiếu trong bảng lookup `Face` và `Suit`.

Hàm `fill()` dùng khởi tạo các lá bài trong mảng `deck`. Khi gán nước cho lá bài thứ `i` (`i ∈ [0, 52)`) ta dùng công thức `i % 13` để nước của lá bài luôn thuộc `[0, 13)`. Khi gán chất cho lá bài ta cũng có thể làm tương tự với công thức `i % 4`, nhưng nên dùng công thức `i / 13` hơn vì:

- Vẫn bảo đảm chất của lá bài `i / 13 ∈ [0, 4)` do `i ∈ [0, 52)`.
- Chất sẽ được gán giống nhau cho từng bộ 13 lá bài. Do 13 không là bội số của 4 nên dùng công thức `i % 4` được, nếu không sẽ có là bài cùng nước cùng chất.

Hàm `shuffle()` dùng trộn bài ngẫu nhiên. Thao tác này được tiến hành như sau: hoán chuyển từng lá bài cần trộn với một lá bài ngẫu nhiên trong 52 lá bài, nghĩa là có thể hoán chuyển với chính nó.

Hàm `deal()` dùng chia bài, mô phỏng gần với thực tế giống các thao tác khác trong bài tập, thao tác này thực hiện như sau: chia từng đợt 4 lá bài cho 4 người chơi theo đúng thứ tự, mỗi người một lá bài.

Bài 171: (trang 49)

```
#include <stdio.h>

union charbit {
  char c;
  struct {
    unsigned b0:1, b1:1, b2:1, b3:1,
             b4:1, b5:1, b6:1, b7:1;
  } b;
};

int main() {
  union charbit n;

  printf( "Nhap 1 ky tu: " );
  scanf( "%c", &n.c );
  printf( "%u %u %u %u %u %u %u %u\n",
          n.b.b7, n.b.b6, n.b.b5, n.b.b4,
          n.b.b3, n.b.b2, n.b.b1, n.b.b0 );
  return 0;
}
```

Ta dùng kỹ thuật "dual view", xem dữ liệu từ các hướng khác nhau với sự hỗ trợ của hai kiểu dữ liệu:

- `union` là một kiểu dữ liệu đặc biệt, tại những thời điểm khác nhau, có thể lưu giữ các đối tượng có kiểu và kích thước khác nhau.

Khác với structure, `union` khai báo nhiều kiểu dữ liệu, nhưng trong *một thời điểm* chỉ có duy nhất *một trị* được lưu với một trong các kiểu dữ liệu khai báo trong `union`.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Các thành viên của structure nối tiếp nhau trong vùng nhớ cấp cho structure, còn các thành viên của union chồng lấp lên nhau trong cùng một vùng nhớ cấp cho `union`, có kích thước bằng kích thước của kiểu dữ liệu lớn nhất được khai báo trong `union`.
Vì trình biên dịch không thể xác định ta lưu dữ liệu kiểu nào vào `union` hoặc đọc dữ liệu từ `union` với kiểu nào; nên cần phải nhớ kiểu của dữ liệu hiện được `union` lưu trữ. Dữ liệu được truy xuất từ `union` thông qua toán tử truy xuất thành viên `"."` hoặc `"->"`, giống như với structure.
Một tiện ích khi dùng `union` là ghi dữ liệu vào `union` với một kiểu và đọc chính dữ liệu đó từ `union` với một kiểu khác, cả hai kiểu này đều khai báo trong `union`. Bài tập trên dùng cách này để "diễn dịch" (interpreted) dữ liệu: ghi dữ liệu vào `union` với kiểu `char`, sau đó đọc chính dữ liệu đó từ `union` với kiểu structure chứa các `bit field`.
- Một `bit field` là một biến nguyên chứa một số chỉ định các bit. `bit field` giúp lưu trữ mẩu tin dưới dạng nén gọn nhất. Ví dụ:
structure `Date` chưa nén:

```
struct Date {
  unsigned month;
  unsigned day;
  int year;
};
```

structure `Date` đã nén bằng cách dùng `bit field`, kích thước chỉ còn 4 byte:

```
struct Date {
  unsigned month:  4; /* lưu được 24 tháng */              .
  unsigned day  :  5; /* lưu được 25 ngày */
  int year      : 22; /* lưu được 222 năm (-2097152 đến +2097151) */
};
```

Ngoài cách dùng các toán tử bitwise, người ta dùng `bit field` như là một cách thao tác đến từng bit cụ thể *theo tên*.
Trong bài tập trên, dùng `union` trị `char` được "diễn dịch" thành structure các `bit field`. Sau đó ta dùng toán tử truy xuất thành viên để thao tác đến từng bit cụ thể theo tên khai báo của chúng trong các `bit field`.

Bài 172: (trang 49)

```
#include <stdio.h>

long filesize( char* filename ) {
  long length = -1L;
  FILE* f;
  f = fopen( filename, "rb" );
  if ( f ) {
    fseek( f, 0L, SEEK_END );
    length = ftell( f );
    fclose( f );
  }
  return length;
}

int main( int argc, char *argv[] ) {
  while ( --argc ) {
    char* fname = *(++argv);
    long len = filesize( fname );
    if ( len == -1L )
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
      printf( "%s [unknown]\n", fname );
    else
      printf( "%s [%ld byte(s)]\n", fname, len );
  }
  return 0;
}
```

Có hai cơ chế di chuyển đến vị trí chỉ định trong một tập tin nhị phân.

- Cách kinh điển là dùng hàm:

`int fseek( FILE* stream, long offset, int origin );`

Hàm này dùng thay đổi vị trí của "con trỏ nội" truy xuất tập tin (file position indicator) liên kết với *stream*, tùy theo trị của *offset* và *origin*. Vị trí mới là một đoạn dài *offset* byte kể từ vị trí được chỉ định bởi *origin*. *origin* phải là một trong các macro sau (thuộc `stdio.h`):

`SEEK_SET`	điểm bắt đầu tập tin
`SEEK_CUR`	điểm hiện tại của "con trỏ nội"
`SEEK_END`	điểm kết thúc tập tin

offset phải là một số `long`, có thể dương (di chuyển tới), âm (di chuyển lui), `0` (đứng tại chỗ).

Ví dụ: `fseek( f, 0L, SEEK_END );` "con trỏ nội" chỉ ngay điểm cuối tập tin.

- Hàm `long ftell( FILE* stream );` trả về vị trí hiện tại của "con trỏ nội" như một số `long`. Với stream nhị phân, đó chính là số byte kể từ đầu tập tin tới "con trỏ nội". Với stream văn bản, hàm hoạt động không chính xác.

Như vậy, trước hết ta di chuyển "con trỏ nội" đến điểm cuối tập tin bằng `fseek()`, rồi dùng `ftell()` để lấy vị trí của nó, đây cũng chính là kích thước của tập tin. Chú ý là stream nối với tập tin phải được mở ở chế độ nhị phân (`"rb"` : chỉ đọc (read) và nhị phân (binary)), để `ftell()` hoạt động chính xác.

Do dùng kiểu `long` nên `fseek()` và `ftell()` có thể bị hạn chế bởi kích thước tập tin, ta có thể thay thế bằng cách dùng cặp hàm:

```
int fgetpos( FILE* stream, fpos_t* position );
int fsetpos( FILE* stream, const fpos_t* position );
```

Ví dụ dùng `fgetpos()`:

```
fpos_t filesize( char* filename ) {
  fpos_t length = -1;
  FILE* f = fopen( filename, "rb" );
  if ( f ) {
    fseek( f, 0L, SEEK_END );
    fgetpos( f, &length );
    fclose( f );
  }
  return length;
}
```

Bài 173: (trang 49)

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

int main() {
  FILE *file1, *file2;
  int i;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  double r;

  srand( time( NULL ) );
  file1 = fopen( "INTEGER.DAT", "w" );
  if ( !file1 ) { perror( "Loi!" ); return 1; }
  for ( i = 0; i < 5; ++i )
    fprintf( file1, "%d\n", rand() );
  fclose( file1 );

  file2 = fopen( "REAL.DAT", "w" );
  if ( !file2 ) { perror( "Loi!" ); return 1; }
  for ( i = 0; i < 5; ++i )
    fprintf( file2, "%f\n", rand() / ( double) RAND_MAX );
  fclose ( file2 );
  printf( "Ghi xong file...\n" );

  file1 = fopen( "INTEGER.DAT", "r" );
  if ( !file1 ) { perror( "Loi!" ); return 1; }
  while ( fscanf( file1, "%d", &i ) != EOF )
    printf( "%8d ", i );
  putchar( '\n' );
  fclose( file1 );

  file2 = fopen( "REAL.DAT", "r" );
  if ( !file2 ) { perror( "Loi!" ); return 1; }
  while ( fscanf ( file2, "%lf", &r ) != EOF )
    printf( "%8.5f ", r );
  putchar( '\n' );
  fclose( file2 );
  printf( "Doc xong file...\n" );
  return 0;
}
```

Bài tập này ôn tập thao tác nhập xuất có định dạng vào stream văn bản với sự hỗ trợ của các hàm sau:

- Hàm `fprintf()` giống với hàm `printf()`, ngoại trừ tham số đầu tiên của nó là stream xuất mà ta sẽ ghi đến. Ví dụ: để ghi một số dữ liệu ra thiết bị xuất chuẩn (`stdout`, mặc định liên kết với màn hình), ta dùng:

```
printf( "%d %.1f %-39s", n, flt, word );
```

Tương tự, để ghi chúng vào stream:

```
fprintf( out_stream, "%d %.1f %-39s", n, flt, word );
```

`stdout` cũng được xem như tập tin nên có thể dùng `fprintf()` như `printf()`:

```
fprintf( stdout, "%d %.1f %-39s", n, flt, word );
```

- Hàm `fscanf()` giống với hàm `scanf()`, ngoại trừ tham số đầu tiên của nó là stream nhập mà ta sẽ nhận dữ liệu. Ví dụ: để đọc vào một số dữ liệu từ thiết bị nhập chuẩn (`stdin`, mặc định liên kết với bàn phím), ta dùng:

```
scanf( "%d %f %39s", &n, &flt, word );
```

Tương tự, để đọc chúng từ stream:

```
fscanf( in_stream, "%d %f %39s", &n, &flt, word );
```

`stdin` cũng được xem như tập tin nên có thể dùng `fscanf()` như `scanf()`:

```
fscanf( stdin, "%d %f %39s", &n, &flt, word );
```

Nếu `fscanf()` đọc đến cuối tập tin, nó sẽ trả về trị `EOF`.

Bài 174: (trang 49)

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
#include <stdio.h>

int main() {
  FILE* f;
  unsigned code;
  char name[32];
  char add[32];
  char birthday[10];

  f = fopen( "PERSON.DAT", "r" );
  if ( !f ) { perror( "Loi!" ); return 1; }
  while ( ( fscanf( f, "%u:%32[^,],%32[^:]:%10[^\n]",
            &code, name, add, birthday ) ) != EOF ) {
    printf( "%-32s[code: %u]\n", name, code );
    printf( "  Address : [%s]\n", add );
    printf( "  Birthday: [%s]\n", birthday );
  }
  fclose( f );
  return 0;
}
```

Cũng giống như hàm `scanf()`, `fscanf()` dùng các chuỗi định dạng, gọi chính xác hơn là các đặc tả chuyển đổi (conversion specification), giúp xử lý chuyển đổi rất linh hoạt dữ liệu nhập. Cú pháp như sau:

`%[*][field_width][length_modifier]specifier`

- Tùy chọn *`field_width`* là một số nguyên dương chỉ định số tối đa các ký tự được đọc và chuyển đổi. Ví dụ: `scanf( "%32s", s );` sẽ nhận vào `s` tối đa 32 ký tự (kể cả ký tự space nếu có trong dữ liệu nhập.
- Tùy chọn *`length_modifier`*, tùy theo *`specifier`* tương ứng, mô tả chi tiết thêm *`specifier`* đó. Ví dụ: `%lf` gồm *`length_modifier`* `l` và *`specifier`* `f` xử lý dữ liệu nhập thành một số `double`.
- *`specifier`* chỉ định kiểu tham số và cách dữ liệu nhập được xử lý chuyển đổi. Mỗi *`specifier`* có một tham số tương ứng. Một vài *`specifier`* ít được mô tả trong các tài liệu tiếng Việt:

specifier	Kiểu tham số	Mô tả
`[scanset]`	`char *`	Tập ký tự dùng chuyển đổi. Xem bên dưới.

`n`	`int *`	Không có dữ liệu nhập, `scanf()` lưu số ký tự đã đọc được cho đến lúc đó từ ngõ nhập vào tham số tương ứng. Ví dụ:
		`scanf( "%s%f%n%i", s, &x, &d, &y );`
		`printf( "[%s][%g][%d][%i]\n", s, x, d, y );`
		nhập `abc 3.14 100`
		xuất `[abc][3.14][8][100]`

`*c`	Không có	Một dấu `%` đơn, không lưu trị nhập.

Specifier `[...]` tương đương với `s`, ngoài ra nó còn so trùng dữ liệu nhập với tập ký tự giữa hai dấu `[]` gọi là *scanset* (có thể có hoặc không ký tự khoảng trắng, có thể có cả ký tự `[` và `]`), nghĩa là `scanf()` sẽ ngưng chuyển đổi dữ liệu nhập khi có ký tự nhập *không thuộc* scanset.

Ví dụ: chỉ nhập vào `s` các ký tự số, ngưng nhập ngay khi phát hiện ký tự khác số.

```
scanf( "%[0-9]s", s );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Nếu ký tự đầu tiên của scanset là ^, sẽ ngưng chuyển đổi dữ liệu nhập khi gặp ký tự *thuộc* scanset.
Ví dụ: nhập 10 ký tự vào s, ngưng nhập khi phát hiện dấu hai chấm.

```
scanf( "%10[^:]s", s );
```

Chuỗi `%*specifier` sẽ đọc dữ liệu chỉ định bởi *specifier* nhưng bỏ qua không gán cho tham số nào.
Ví dụ: nhập `10/20`, sẽ đặt trị `10` vào x, bỏ qua dấu `/`, đặt trị `20` vào y.

```
scanf( "%d%*c%d", &x, &y );
```

Bài tập: viết hàm `update()`, dùng cập nhật cho một structure `Student`. Với những trường không cần cập nhật, có thể nhấn phím `Enter` để bỏ qua.

```
typedef struct {
  char name[20];
  int age;
  float mark;
} Student;

void update( Student *student ) {
  char _name[20], buf, *t;
  int _age, size;
  float _mark;

  printf( "Nhap [Enter] de bo qua\n" );
  printf( "Ten moi? " );
  fgets( _name, 20, stdin );
  if ( ( t = strrchr( _name, '\n' ) ) != NULL ) *t = '\0';
  if ( strlen( _name ) > 0 ) strcpy( student->name, _name );

  printf( "Tuoi moi? " );
  while ( scanf( "%1[\n]", &buf ) == 0 ) {
    if ( scanf( "%d", &_age ) == 1 ) student->age = _age;
    scanf( "%*[^\n]" );
  }

  printf( "Diem moi? " );
  while ( scanf( "%1[\n]", &buf ) == 0 ) {
    if ( scanf( "%f", &_mark ) == 1 ) student->mark = _mark;
    scanf( "%*[^\n]" );
  }
  printf( "Cap nhat xong...\n" );
}
```

Bài 175: (trang 50)

```
#include <stdio.h>
#define UPPER( a ) (a) = ((a)>='a' && (a)<='z')? (a)-32:(a)

int main( int argc, char* argv[] ) {
  FILE *fin, *fout;
  int ch;

  if ( argc != 3 ) {
    printf( "Cu phap: UPPER lowerfile upperfile\n" );
    return 0;
  }
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  if ( ( fin = fopen( argv[1], "r" ) ) == NULL ) {
    perror( "Loi mo file nhap" );
    return 1;
  } else if ( ( fout = fopen(argv[2], "w") ) == NULL ) {
    perror( "Loi mo file xuat" );
    fclose(fin);
    return 1;
  }

  while ( ( ch = fgetc( fin ) ) != EOF )
    fputc( UPPER( ch ), fout );

  fclose( fin );
  fclose( fout );
  printf( "Chuyen thanh chu hoa xong...\n" );
  return 0;
}
```

Để làm việc với từng ký tự trong một tập tin, dùng các hàm sau:
- Hàm `int fgetc( FILE* stream )`: lấy ký tự kế tiếp từ *stream* chỉ định và tăng một "con trỏ nội" dùng truy xuất tập tin. Ký tự được đọc như một `unsigned char` và được chuyển đổi thành `int`. Nếu đến cuối tập tin hoặc gặp lỗi hàm `fgetc()` trả về `EOF`. Với tập tin nhị phân, phải dùng `feof()` để kiểm tra cuối tập tin.
- Hàm `int fputc( int ch, FILE *stream )`: ghi ký tự *ch* đến *stream* chỉ định rồi tăng "con trỏ nội" dùng truy xuất tập tin. Ký tự nhập sẽ được chuyển đổi thành `unsigned char`. Hàm `fputc()` trả về trị của ký tự đã ghi hoặc `EOF` nếu gặp lỗi. Với tập tin nhị phân, phải dùng `ferror()` để xác định có lỗi xảy ra.
- `getc()` tương tự `fgetc()` và `putc()` tương tự `fputc()` nhưng là các macro.

Bài 176: (trang 50)

```
#include <stdio.h>

int main( int argc, char* argv[] ) {
  FILE *fin, *fout;
  int ch;

  if ( argc != 3 ) {
    printf( "Cu phap: CIPHER originalfile encodefile\n" );
    printf( "hoac   : CIPHER encodefile originalfile\n" );
    return 0;
  }

  if ( ( fin = fopen( argv[1], "r" ) ) == NULL ) {
    perror( "Loi mo file nhap" );
    return 1;
  } else if ( ( fout = fopen( argv[2], "w" ) ) == NULL ) {
    perror( "Loi mo file xuat" );
    fclose( fin );
    return 1;
  }

  while ( ( ch = fgetc( fin ) ) != EOF )
    fputc( ( ch + 128 ) % 256, fout );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  fclose( fin );
  fclose( fout );
  printf( "Xu ly xong...\n" );
  return 0;
}
```

Dùng các hàm `fgetc()` và `fputc()`, giống bài 175 (trang 256). Đây là mã hóa đối xứng nên chương trình trên có thể dùng để mã hóa lẫn giải mã.

Bài 177: (trang 50)

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

int main( int argc, char* argv[] ) {
  FILE *fin;
  int ch, pos;

  if ( argc != 3 ) {
    printf( "Cu phap: TESTFILE <options> filename\n" );
    printf( "Options:\n"
        "  n  sinh n ky tu ngau nhien, dat vao 'filename'\n"
        " -r  hien thi noi dung 'filename'\n"
        " -v  hien thi noi dung 'filename', thu tu nguoc\n" );
    return 0;
  }

  /* Tạo file chứa ký tự ngẫu nhiên: n filename */
  if ( argv[1][0] != '-' ) {
    srand( time( NULL ) );
    FILE* fout = fopen( argv[2], "w" );
    if ( fout == NULL ) {
      perror( "Loi" );
      return 1;
    }
    for ( int i = 0; i < atoi( argv[1] ); ++i )
      fputc( rand() % 26 + 'A', fout );
    fclose( fout );
    printf( "File da duoc tao...\n" );
  }
  /* Hiển thị nội dung file: -v filename */
  else if ( !strcmp( argv[1], "-v" ) ) {
    if ( ( fin = fopen( argv[2], "r" ) ) == NULL ) {
      perror( "Loi" );
      return 1;
    }
    while ( ( ch = fgetc( fin ) ) != EOF ) {
      fputc( ch, stdout );
      fputc( ' ', stdout );
    }
    fclose( fin );
    printf( "\nCuoi file...\n" );
  }
  /* Hiển thị nội dung file theo thứ tự ngược lại: -r filename */
  else if ( !strcmp( argv[1],"-r" ) ) {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    pos = -1;
    if ( ( fin = fopen( argv[2], "r" ) ) == NULL ) {
      perror( "Loi" );
      return 1;
    }
    do {
      fseek( fin, 1L * pos, SEEK_END );
      if ( ( ch = fgetc( fin ) ) != EOF ) {
        fputc( ch, stdout );
        fputc( ' ', stdout );
      }
      pos--;
    } while ( ch != EOF );
    fclose( fin );
    printf( "\nDau file...\n" );
  } else {
    printf( "Tham so khong hop le\n" );
    return 2;
  }
  return 0;
}
```

Sau khi xử lý các tùy chọn dòng lệnh, chương trình sẽ lựa chọn thực thi một trong các công việc sau:

- Tạo tập tin chứa các ký tự ngẫu nhiên: tùy chọn dòng lệnh được chuyển từ chuỗi thành số `int` bằng hàm `atoi()`. Hàm `fputc()` ghi từng ký tự vào stream, các ký tự lấy ngẫu nhiên trong đoạn `[0, 16) + 'A'` (bảng alphabet chữ hoa).
- Hiển thị nội dung tập tin: khi mở stream xuất, "con trỏ nội" truy xuất tập tin nằm ngay đầu tập tin. Dùng `fgetc()` lấy lần lượt từng ký tự và dùng `fputc()` xuất ra màn hình. Chú ý, các lệnh sau đây là tương đương:

```
fputc( ch, stdout );
putc( ch, stdout );
putchar( ch );
```

- Hiển thị nội dung tập tin theo thứ tự ngược: sau khi mở stream xuất, dùng `fseek()` định vị "con trỏ nội" truy xuất tập tin chỉ cuối tập tin (*không phải* chỉ ký tự cuối tập tin). Để đi ngược lên đầu tập tin qua từng ký tự, xem bài 172 (trang 252), offset `pos` phải là số âm và giảm dần từng đơn vị.

Bài 178: (trang 50)

```
#include <stdio.h>

int main() {
  int c, nlines = 0, f = 1;

  while ( ( c = getchar() )!= EOF ) {
    if ( f ) {
      printf( "%3u: ", ++nlines );
      f = 0;
    }
    putchar( c );
    if ( c == '\n' ) f = 1;
  }
  return 0;
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Khi chương trình C chạy, có 3 stream mở sẵn liên kết với nó:
- stream nhập chuẩn `stdin`, liên kết với bàn phím. Ta có thể định hướng lại stream này bằng cách dùng filter (bộ lọc) hay pipe (đường ống). Hệ điều hành thường hỗ trợ các tác vụ này:

`lines < p1.c` dùng pipe, `stdin` của `lines` là `p1.c`

`type p1.c | lines` dùng filter, `stdin` của `lines` là kết quả lệnh `type`

Các lệnh sau đây đều lấy một ký tự từ stream nhập chuẩn (`stdin`) đưa vào `ch`:

```
ch = fgetc( stdin );
ch = getc( ch, stdin );
ch = getchar();
```

- stream xuất chuẩn `stdout`, liên kết với màn hình. Ta có thể định hướng lại stream này để kết quả xuất đổ vào một tập tin khác.

`type p1.c > p2.c` `stdout` của lệnh `type` là `p2.c`

`lines < p1.c > p2.c` `stdout` của `lines` là `p2.c`

Các lệnh sau đây đều đặt ký tự `ch` vào stream xuất chuẩn (`stdout`):

```
fputc( ch, stdout );
putc( ch, stdout );
putchar( ch );
```

- stream lỗi chuẩn `stderr`.

Chương trình hoạt động như sau:

`stdin` (bàn phím, tập tin nguồn) ⇒ `ch = getchar()`

⇒ nếu `ch = '\n'` chèn số dòng (`unsigned int`) vào stream

⇒ `putchar( ch )` ⇒ `stdout` (màn hình, tập tin nguồn khác)

Bài tập: Nhập từ bàn phím cho đến khi nhập ký tự `#`. Đếm số [space], [enter] và các ký tự khác (kể cả ký tự `#`).

```
#include <stdio.h>

int main() {
  char c;
  unsigned space, ret, other;
  space = ret = other = 0;
  do {
    c = getchar();
    switch ( c ) {
      case '\n': ret++; break;
      case ' ': space++; break;
      default: other++;
    }
  } while ( c != '#' );
  printf( "[space]: %u, [return]: %u, [other]: %u \n", space, ret, other );
  return 0;
}
```

Bài 179: (trang 51)

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

typedef struct {
  int code;
  char name[20];
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  double avgmark;
} STUDENT;

void createFile( char *s ) {
  FILE *f;
  STUDENT r;
  char c;
  printf( "GHI FILE\n" );
  f = fopen( s, "a" );
  if ( !f ) fprintf( stderr, "Loi tao file\n" );
  else {
    do {
      do {
        c = 0;
        printf( "Nhap mot mau tin( y/n )? " );
        scanf( "%1[yn]c", &c );
        while ( getchar() != '\n' ) { }
      } while ( !c );
      if ( c != 'n' ) {
        printf( " Ma so   > " );
        scanf( "%d", &r.code );
        while ( getchar() != '\n' ) { }
        printf( " Ten     > " );
        fgets( r.name, 20, stdin );
        char* t = strrchr( r.name, '\n' );
        if ( t != NULL ) *t = '\0';
        printf( " Diem TB > " );
        scanf( "%lf", &r.avgmark );
        while ( getchar() != '\n' ) { }
        fwrite( &r, sizeof( STUDENT ), 1, f );
      }
    } while ( c != 'n' );
    fclose( f );
    printf( "Da ghi file...\n" );
  }
}

void readFile( char *s ) {
  FILE *f;
  STUDENT r;
  printf( "DOC FILE\n" );
  f = fopen( s, "r" );
  if ( !f ) fprintf( stderr, "Loi mo file\n" );
  else {
    int count = 0;
    while( ( fread( &r, sizeof( STUDENT ), 1, f ) ) != 0 )
    printf( "%-3d%-5d%-16s%-5g\n", ++count, r.code, r.name, r.avgmark );
    printf( "Tong cong: %d record(s)\n", count );
    fclose( f );
  }
}

long find( FILE *f, int x ) {
  STUDENT r;
  int n;
  do {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    long d = ftell( f );
    n = fread( &r, sizeof( STUDENT ), 1, f );
    if ( x == r.code ) return d;
  } while ( n );
  return -1;
}

void adjustFile( char *s ) {
  FILE *f;
  STUDENT r;
  int x;
  printf( "SUA\n" );
  f = fopen( s, "r+b" );
  if ( !f ) fprintf( stderr, "Loi mo file\n" );
  else {
    printf( "Ma so > ");
    scanf( "%d", &x );
    long d = find( f, x );
    if ( d > -1 ) {
      fseek( f, d, SEEK_SET );
      fread( &r, sizeof( STUDENT ), 1, f );
      printf( "  %s\n", r.name );
      printf( "Diem moi > " );
      scanf( "%lf", &r.avgmark );
      while ( getchar() != '\n' ) { }
      fseek( f, d, SEEK_SET );
      if ( fwrite( &r, sizeof( STUDENT ), 1, f ) )
        printf( "Da cap nhat...\n");
    }
    else printf( "Khong tim thay...\n" );
    fclose( f );
  }
}

int main() {
  char c;
  char s[20];

  printf( "Ten file? " );
  scanf( "%19s", &s );
  while ( getchar() != '\n' ) { }
  if ( *s == '\0' ) strcpy( s, "DEFAULT.DAT" );
  do {
    printf( "MENU (File '%s')\n", s );
    printf( "----\n"
            "[1]. Them\n"
            "[2]. Doc\n"
            "[3]. Sua\n"
            "[4]. Thoat\n");
    printf( "Chon tac vu: " );
    c = getchar();
    while ( getchar() != '\n' ) { }
    switch ( c - 48 ) {
      case 1: createFile( s ); break;
      case 2: readFile( s ); break;
      case 3: adjustFile( s );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    }
  } while ( c != '4' );
  printf( "Bye...\n" );
  return 0;
}
```

Khi đọc ghi các structure vào một tập tin, tốt nhất ta xem tập tin đó như tập tin nhị phân. Như vậy, đọc ghi một structure giống như đọc ghi một khối byte có kích thước tương ứng, bằng cách dùng phối hợp các hàm:

- Hàm `size_t fread( void *buf, size_t size, size_t count, FILE *stream );` đọc *count* đối tượng, mỗi đối tượng có kích thước *size* byte, từ *stream* và lưu nó vào mảng do *buf* chỉ đến. Sau đó, "con trỏ nội" truy xuất tập tin sẽ di chuyển một đoạn bằng số ký tự đọc được, `fread()` cũng trả về số ký tự này.
- Hàm `size_t fwrite( const void *buf, size_t size, size_t count, FILE *stream );` ghi *count* đối tượng, mỗi đối tượng có kích thước *size* byte, từ mảng do *buf* chỉ đến vào *stream*. Sau đó, "con trỏ nội" truy xuất tập tin di chuyển một đoạn bằng số ký tự được ghi, `fwrite()` trả về số ký tự thực sự ghi thành công.

Để di chuyển trong tập tin nhị phân, dùng các hàm: `fseek()`, `ftell()`, `rewind()`.

Các thao tác trên structure đã trình bày trong phần bài tập về structure. Bài tập này chỉ tập trung vào thao tác đọc ghi một structure vào tập tin:

- Tạo tập tin (hoặc ghi thêm): tập tin với tên nhập vào (hoặc tên mặc định `DEFAULT.DAT`) được mở với chế độ ghi tiếp (`"a"` - append). Từng structure sau khi nhập đủ thông tin sẽ được ghi như một khối byte vào tập tin bằng hàm `fwrite()`.
- Đọc tập tin: tập tin được mở với chế độ đọc (`"r"` - read). Lần lượt từng khối byte tương ứng với một structure được đọc vào structure tạm bằng hàm `fread()`. Sau đó thông tin được trích từ structure tạm này để xuất ra màn hình.
- Hiệu chỉnh tập tin: tập tin được mở với chế độ đọc *có cập nhật* (`"r+"`). Vị trí của structure chứa thông tin cần hiệu chỉnh do hàm `find()` của chúng ta cung cấp. Ta thực hiện một chuỗi thao tác lần lượt theo thứ tự sau:

  Di chuyển đến vị trí này bằng hàm `fseek()`.
  Đọc khối byte tương ứng với một structure ra structure tạm bằng `fread()`.
  Hiệu chỉnh dữ liệu trên structure tạm.
  Lại di chuyển đến vị trí của structure đó trong tập tin bằng hàm `fseek()`.
  Ghi structure tạm (đã hiệu chỉnh) vào tập tin bằng `fwrite()`.
- Hàm `find()`: đọc lần lượt từng khối byte tương ứng với một structure ra structure tạm rồi kiểm soát thành viên `code` của chúng để tìm được structure cần tìm. Đồng thời dùng `ftell()` lưu vị trí tại đầu các khối byte tương ứng với structure *đang đọc*. Vị trí này sẽ được trả về nếu phát hiện ra structure cần tìm.

Cần kiểm soát kỹ buffer nhập, súc stream nhập để tránh sót ký tự `'\n'` khi nhập số hoặc khi nhận một ký tự bằng `getchar()`:

```
while ( getchar() != '\n' ) { }
```

Bạn cần tránh lỗi phổ biến: để sót ký tự `'\n'` trong buffer nhập khi gọi một module của chương trình và module tiếp theo chịu ảnh hưởng làm chương trình chạy không theo điều khiển.

Bạn cũng nên học tập cách tạo menu đơn giản như trong bài.

Bài 180: (trang 52)

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

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
#include <string.h>

typedef struct {
  char w[8];          /* từ được lưu (chữ thường) */
  int freq;           /* tần số xuất hiện từ */
} WORD;

int main() {
  WORD words[1000];
  char filename[20];
  char s[255];
  char* delimiter = " ,;.:?!()\"\\'\n";
  FILE *fin;
  int i, k, nlines, nwords;

  do {
    printf( "Nhap ten file: " );
    scanf( "%19s", filename );
    fin = fopen( filename, "r" );
    if ( !fin )
      printf( "Khong tim thay file: %s\n", filename );
  } while ( !fin );

  nlines = nwords = k = 0;
  while ( fgets( s, sizeof( s ), fin ) != NULL ) {
    nlines++;
    char* p = s;
    while ( ( p = strtok( p, delimiter ) ) != NULL ) {
      p = strlwr( p );
      for ( i = 0; i < k; ++i )
      if ( ( strcmp( words[i].w, p ) ) == 0 )
        { words[i].freq++; break; }
      if ( i == k ) {
        strcpy( words[k].w, p );
        words[k++].freq = 1;
      }
      nwords++;
      p = NULL;
    }
  }
  fclose( fin );

  printf( "File %s co %d dong, %d tu, voi tan so cac tu:\n",
          filename, nlines, nwords );
  for ( i = 0; i < k; ++i ) {
    printf( "%8s:%3d ", words[i].w, words[i].freq );
    if ( i > 0 && (i + 1) % 4 == 0 ) putchar( '\n' );
  }
  putchar( '\n' );
  return 0;
}
```

Thao tác chủ yếu của bài tập trên là *đọc một dòng văn bản từ tập tin* (xem như mở trong chế độ văn bản) bằng hàm `fgets()`. Chú ý, chuỗi do `fgets()` nhận vào có cả ký tự `'\n'`, cần liệt kê trong tập delimiter để `strtok()` lọc ra.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Dòng văn bản sau khi đọc vào chuỗi thì tiến hành tách từ bằng `strtok()`, dùng "vòng lặp xử lý chuỗi điển hình", xem bài 118 (trang 190).
Mỗi từ tách ra sẽ được kiểm tra xem đã có trong mảng `words` hay không (so sánh bằng `strcmp()`). Mảng này lưu các structure `WORD`, bao gồm từ được lưu (chữ thường) và tần số xuất hiện của từ đó.

Bài 181: (trang 52)

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

int main() {
  FILE* fin;
  char buf[255];
  char s[30];
  char* p;
  int nline = 0;

  do {
    printf( "Nhap ten file: " );
    scanf( "%29s", s );
    while ( getchar() != '\n' ) { }
    fin = fopen( s, "r" );
    if ( !fin ) printf( "Khong tim thay file: %s\n", s );
  } while ( !fin );

  printf( "Nhap chuoi tim: " );
  fgets( s, 30, stdin );
  if ( ( p = strrchr( s, '\n' ) ) != NULL ) *p = '\0';

  while ( fgets( buf, sizeof( buf ), fin ) != NULL ) {
    if ( ( strstr( buf, s ) ) != NULL ) {
      printf( "Dong %d: ", nline );
      p = buf;
      while ( ( p = strstr( p, s ) ) != NULL ) {
        printf( "%d ", p - buf );
        p++;
      }
      putchar( '\n' );
    }
    nline++;
  }
  fclose( fin );
  return 0;
}
```

Tương tự bài 180 (trang 263), từng dòng của tập tin văn bản cũng được đọc nhờ hàm `fgets()`. Sau khi đọc dòng văn bản vào chuỗi, ta tiến hành tìm vị trí chuỗi con bằng `strstr()`, dùng "vòng lặp xử lý chuỗi điển hình", tương tự như bài 116 (trang 186).

Bài 182: (trang 53)

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

typedef struct {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  char name[15];
  char tel[15];
  char add[25];
} PERSON;

void createBinFile( char *s ) {
  FILE *f;
  PERSON r;
  char c;
  printf( "Tao file nhi phan %s...\n", s );
  f = fopen( s, "w" );
  if ( !f ) fprintf( stderr, "Loi tao file\n" );
  else {
    do {
      do {
        c = 0;
        printf( "Nhap mot mau tin( y/n )? " );
        scanf( "%1[yn]c", &c );
        while ( getchar() != '\n' ) { }
      } while (!c);
      if ( c != 'n' ) {
        char *t;
        printf( " Ten         > " );
        fgets( r.name, 15, stdin );
        if ( ( t = strrchr( r.name, '\n' ) ) != NULL ) *t = '\0';
        printf( " Dien thoai > " );
        fgets( r.tel, 15, stdin );
        if ( ( t = strrchr( r.tel, '\n' ) ) != NULL ) *t = '\0';
        printf( " Dia chi    > " );
        fgets( r.add, 25, stdin );
        if ( ( t = strrchr( r.add, '\n' ) ) != NULL ) *t = '\0';
        fwrite( &r, sizeof( PERSON ), 1, f );
      }
    } while ( c != 'n' );
    fclose( f );
  }
}

void createTextFile( char *s, char *s1 ) {
  FILE *f, *f1;
  PERSON r;
  printf( "Tao file van ban %s...\n", s );
  f1 = fopen( s1, "r" );
  f = fopen( s, "w" );
  if ( !f1 || !f ) fprintf( stderr, "Loi mo file\n" );
  else
    while ( ( fread( &r, sizeof( PERSON ), 1, f1 ) ) != 0 )
      fprintf( f, "%-15s%-15s%-25s\n", r.name, r.tel, r.add );
  fclose( f ) ;
  fclose( f1 );
}

void displayTextFile( char *s ) {
  FILE *f;
  printf( "Hien thi file van ban %s...\n", s );
  f = fopen( s, "r" );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  if ( !f ) fprintf( stderr, "Loi mo file\n" );
  else {
    char buf[100];
    while ( fgets( buf, sizeof( buf ), f ) != NULL )
      fprintf( stdout, "%s", buf );
  }
  fclose(f);
}

int main() {
  createBinFile( "PERSON.DAT" );
  createTextFile( "PERSON.TXT", "PERSON.DAT" );
  displayTextFile( "PERSON.TXT" );
  return 0;
}
```

Bài tập trên giúp nắm vững và phân biệt các hàm xử lý tập tin trong cả hai trường hợp dùng stream nhị phân và stream văn bản.
Các yêu cầu của bài tập được thực hiện bằng các hàm sau:
- Hàm `createBinFile()`: việc tạo một tập tin nhị phân lưu các structure đã được trình bày chi tiết trong bài 179 (trang 260). Ta dùng hàm `fwrite()` để lưu các khối byte tương ứng với từng structure vào tập tin nhị phân.
- Hàm `createTextFile()`: mỗi khối byte tương ứng với một structure được đọc lần lượt từ tập tin nhị phân vào structure tạm bằng hàm `fread()`. Các thông tin trích từ structure tạm được bố trí theo một chuỗi định dạng nhất định và được ghi vào tập tin văn bản bằng hàm `fprintf()`. Hàm `fprintf()` tương tự hàm `printf()`, nhưng ghi kết xuất vào stream thay vì vào ngõ xuất chuẩn `stdout`.
- Hàm `displayTextFile()`: đơn giản đọc từng dòng văn bản trong tập tin văn bản bằng hàm `fgets()` và xuất ra màn hình (dùng `printf()` hoặc `fprintf()`) .

Bài 183: (trang 53)

```
#include <stdio.h>

void reverse ( FILE* fin, FILE* fout ) {
  char buf[255];
  if ( fgets( buf, sizeof( buf ), fin ) == NULL ) return;
  reverse( fin, fout );
  fprintf( fout, "%s", buf );
}

int main( int argc, char* argv[] ) {
  FILE *fin, *fout;

  if ( argc != 3 ) {
    printf( "Cu phap: REVERSE originalfile reversefile\n" );
    return 0;
  }
  if ( ( fin = fopen( argv[1], "r" ) ) == NULL ) {
    perror( "Loi mo file nhap" );
    return 1;
  } else if ( ( fout = fopen( argv[2], "w" ) ) == NULL ) {
    perror( "Loi mo file xuat" );
    fclose( fin );
    return 1;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  }

  reverse( fin, fout );
  fclose( fin );
  fclose( fout );
  printf( "Dao nguoc de quy xong...\n" );
  return 0;
}
```

Cơ sở để gọi đệ quy ở đây là: khi ta dùng hàm `fgets()` đọc một dòng từ tập tin `fin` và dùng hàm `fprintf()` ghi dòng đó vào tập tin `fout`, một "con trỏ nội" truy xuất tập tin *sẽ được duy trì* trong `fin` lẫn trong `fout` *chỉ vị trí truy xuất kế tiếp*, cho đến khi các tập tin này được đóng.
Vì vậy, dù không thay đổi các tham số trong mỗi lời gọi đệ quy, vị trí các "con trỏ nội" cũng tự động thay đổi. Đến lúc "con trỏ nội" của tập tin `fin` chỉ đến cuối, đệ quy sẽ dừng.
Để các dòng đọc từ `fin`, ghi theo thứ tự ngược vào `fout`, ta dùng đệ quy đầu, xem bài 142 (trang 41). Nghĩa là khi đệ quy chấm dứt, các dòng đọc từ `fin` mới được đẩy ra khỏi stack và ghi theo nguyên tắc `LIFO` vào `fout`.

Bài 184: (trang 53)

```
#include <stdio.h>

typedef struct { int dd, mm, yy; } DATE;
typedef struct {
  char name[10];
  DATE bd;
  float salary;
} EMP;

long datecmp( DATE *d1, DATE *d2 ) {
  long t1, t2;
  t1 = ( d1->yy - 1900 ) * 365 + ( d1->mm - 1 ) * 31 + d1->dd;
  t2 = ( d2->yy - 1900 ) * 365 + ( d2->mm - 1 ) * 31 + d2->dd;
  return ( t1 - t2 );
}

int main() {
  EMP e, e1, e2;
  FILE * fp;
  int i, j, n;

  fp = fopen( "EMP.DAT", "wb" );
  if ( !fp ) {
    perror( "Loi" );
    return 1;
  }
  printf( "Nhap so nhan vien: " );
  scanf( "%d", &n );
  while ( getchar() !=  '\n' ) { }
  printf( "Nhap (ten, ngay, thang, nam sinh, luong):\n" );
  for ( i = 0; i < n; ++i ) {
    printf( "%d > ", i + 1 );
    scanf( "%9s%d%d%d%f", e.name, &e.bd.dd, &e.bd.mm, &e.bd.yy, &e.salary );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    fwrite( &e, sizeof( e ), 1, fp );
  }
  fclose ( fp );
  printf( "Nhap du lieu xong ...\n" );

  fp = fopen( "EMP.DAT", "r+b" );
  if ( !fp ) { perror( "Loi" ); return 1; }

  for ( i = n - 1; i > 0; --i ) {
    fread( &e1, sizeof( EMP ), 1, fp );
    for ( j = 0 ; j < i ; ++j ) {
      fread( &e2, sizeof( e2 ), 1, fp );
      if ( datecmp( &e2.bd, &e1.bd ) > 0 ) {
        fseek( fp, -2*(long)sizeof( EMP ), SEEK_CUR );
        fwrite( &e2, sizeof ( EMP ), 1, fp );
        fwrite( &e1, sizeof ( EMP ), 1, fp );
      }
      else e1 = e2;
    }
    rewind ( fp );
  }

  printf ( "Xuat danh sach sap xep: \n" );
  while ( fread ( &e, sizeof( e ), 1, fp ) )
    printf( "%s\t %d/%d/%d \t%7g\n",
          e.name, e.bd.dd, e.bd.mm, e.bd.yy, e.salary );
  fclose( fp );
  return 0;
}
```

Sắp xếp *tăng* theo tuổi có nghĩa là sắp xếp *giảm* theo ngày, tháng, năm sinh. Có thể so sánh ngày tháng bằng hàm `difftime()` của `time.h`. Ở đây, hàm `datecmp()` do chúng ta viết trên không trả về chính xác thời gian chênh lệch giữa hai thời điểm nhưng cũng đủ để so sánh hai thời điểm.

Ta chọn giải thuật sắp xếp kiểu nổi bọt (bubble sort) do đặc điểm so sánh và hoán chuyển dữ liệu trong giải thuật diễn ra giữa hai phần tử kế tiếp nhau. Điều này thuận lợi cho thao tác trên tập tin vì không phải di chuyển nhiều quá bằng `fseek()`. Xem chú giải chi tiết bên dưới.

```
for ( i = n - 1; i > 0; --i ) {
  /* e1 là structure đầu */
  fread( &e1, sizeof( EMP ), 1, fp );
  for ( j = 0 ; j < i ; ++j ) {
    /* tại đây con trỏ nội luôn sau e1, e2 là structure kế tiếp sau e1 */
    fread( &e2, sizeof( e2 ), 1, fp );
    /* so sánh e2 và e1, sắp xếp giảm theo ngày */
    if ( datecmp( &e2.bd, &e1.bd ) > 0 ) {
      /* nếu có sắp xếp, lui 2 structure để ghi e2 rồi e1 */
      fseek( fp, -2*(long)sizeof( EMP ), SEEK_CUR );
      fwrite( &e2, sizeof ( EMP ), 1, fp );
      fwrite ( &e1, sizeof ( EMP ), 1, fp );
      /* sau khi ghi, e2 bây giờ là e1, con trỏ nội ngay sau e1 */
    }
    /* không sắp xếp, dồn e1 thành e2, con trỏ nội ngay sau e1 */
    else e1 = e2;
  }
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  rewind ( fp );    /* trở lại đầu file, chuẩn bị một chuỗi bubble mới */
}
```

Quan trọng nhất trong truy xuất ngẫu nhiên tập tin nhị phân là xác định được chính xác vị trí "con trỏ nội" truy xuất tập tin tại mọi thời điểm.
Sau đây là cách sắp xếp bằng giải thuật sắp xếp kiểu chọn (Selection Sort) dễ hiểu hơn nhưng phải di chuyển trong tập tin bằng `fseek()` nhiều hơn. Xem chú giải chi tiết bên dưới.

```
long d = 0L;
long maxpos;
/* ... */
for ( i = 0; i < n - 1; ++i ) {
  /* d chỉ vị trí structure đang chọn sắp xếp */
  fseek( fp, d, SEEK_SET );
  fread( &e1, sizeof( e1 ), 1, fp );
  for ( j = i + 1 ; j < n ; ++j ) {
    fread( &e2, sizeof( EMP ), 1, fp );
    if ( datecmp( &e2.bd, &e1.bd ) > 0 )
    /* maxpos chỉ vị trí structure có ngày sinh lớn hơn (sắp xếp giảm theo ngày) */
    maxpos = ftell( fp ) - sizeof( EMP );
  }
  /* ghi hoán chuyển hai structure vào file */
  fseek( fp, maxpos, SEEK_SET );
  fwrite ( &e1, sizeof ( EMP ), 1, fp );
  fseek( fp, d, SEEK_SET );
  fwrite ( &e2, sizeof ( EMP ), 1, fp );
  d += sizeof( EMP );
}
rewind( fp );   /* trở lại đầu file để xuất */
```

Bài 185: (trang 54)

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

void createMatrixFile( FILE* fin ) {
  int m, n, i, j;
  printf( "Nhap m, n: " );
  scanf( "%d%d", &m, &n );
  fprintf( fin, "%d %d\n", m, n );
  for ( i = 0; i < m; ++i, fputc( '\n', fin ) )
    for ( j = 0; j < n; ++j )
      fprintf( fin, "%lf ", rand() / ( double )RAND_MAX );
  printf( "Tao xong file chua ma tran...\n" );
}

int mulMatrixFile( FILE* f1, FILE* f2, FILE* f3 ) {
  int m1, n1, m2, n2, i, j, k;
  double *a, *b, t;
  /* dòng m và cột n của ma trận */
  fscanf( f1, "%d%d", &m1, &n1 );
  fscanf( f2, "%d%d", &m2, &n2 );
  if ( n1 != m2 ) { fclose( f1 ); fclose( f2 ); return 2; }

  a = ( double * )calloc( m1 * n1, sizeof( double ) );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  b = ( double * )calloc( m2 * n2, sizeof( double ) );
  if ( !a || !b ) {
    if ( a != NULL ) free( a );
    if ( b != NULL ) free( b );
    return 3;
  }
  /* đọc các phần tử đến mảng một chiều */
  for ( i = 0; i < m1; ++i )
    for ( j = 0; j < n1; ++j )
      fscanf( f1, "%lf", &a[i*n1 + j] );
  for ( i = 0; i < m2; ++i )
    for ( j = 0; j < n2; ++j )
      fscanf( f2, "%lf", &b[i*n2 + j] );

  /* nhân ma trận, dùng công thức ánh xạ mảng hai chiều thành mảng một chiều */
  fprintf( f3, "%d %d\n", m1, n2 );
  for ( i = 0; i < m1; ++i, fputc( '\n', f3 ) )
    for ( j = 0; j < n2; ++j ) {
      for ( t = 0.0, k = 0; k < n1; ++k )
        t += a[i*n1 + k] * b[k*n1 + j];
      fprintf( f3, "%lf ", t );
    }

  free( b );
  free( a );
  printf( "Nhan ma tran da xong...\n" );
  return 0;
}

int main( int argc, char* argv[] ) {
  if ( argc < 3 ) {
    printf( "Cu phap: file1 file2 file3\n" );
    printf( " 2 tham so: tao hai file chua ma tran"
            " so thuc ngau nhien\n"
            " 3 tham so: nhan 2 ma tran trong file1"
            " va file2, dat ket qua vao file3\n" );
    return 0;
  }

  srand( time( NULL ) );
  if ( argc == 3 ) {
    for ( int i = 1; i < argc; ++i ) {
      FILE* fin = fopen( argv[i], "w" );
      if ( !fin ) {
        printf( "Loi tao file\n" );
        return 1;
      }
      createMatrixFile( fin );
      fclose( fin );
    }
  }
  else {
    FILE* f1 = fopen( argv[1], "r" );
    FILE* f2 = fopen( argv[2], "r" );
    FILE* f3 = fopen( argv[3], "w" );
    if ( !f1 || !f2 || !f3 ) {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
      printf( "Loi mo file\n" );
      if ( f1 != NULL ) fclose( f1 );
      if ( f2 != NULL ) fclose( f2 );
      if ( f3 != NULL ) fclose( f3 );
      return 1;
    }
    if ( mulMatrixFile( f1, f2, f3 ) ) {
      printf( "Loi nhan ma tran\n" );
      return 2;
    }
    fclose( f1 );
    fclose( f2 );
    fclose( f3 );
  }
  return 0;
}
```

Bài tập này giúp chúng ta có một kinh nghiệm tốt về các thao tác tương đương giữa việc đọc ghi đến thiết bị chuẩn (`stdin`, `stdout`) với thao tác đọc ghi stream. Các hàm dùng trong bài tập được thiết kế để nhận tham số là con trỏ `FILE`, nhằm làm nổi bật đặc điểm này:

- Hàm `createMatrixFile()` tạo tập tin chứa ma trận: thay vì tạo và xuất thông tin của ma trận (số hàng, số cột, các phần tử) ra màn hình (chuyển nó đến stream `stdout`), ta tạo và chuyển chúng đến một stream đang mở. Nghĩa là ta thay thế các hàm xuất mặc định ra `stdout` thành các hàm xử lý tập tin như:

`putchar( '\n' )` thay bằng `fputc( '\n', fin )`.

`printf( ... )` thay bằng `fprintf( fin, ... )`.

Cần nhấn mạnh, `stdin` và `stdout` cũng là một loại stream. Ví dụ, thay vì tạo ma trận ngẫu nhiên và lưu vào tập tin, ta có thể tạo ma trận ngẫu nhiên và xuất thẳng ra màn hình bằng: `createMatrixFile( stdout );`

```
Nhap m, n: 2 3 ↵
2 3                                dòng và cột ma trận mới tạo
0.001251 0.563585 0.193304         các phần tử ma trận mới tạo
0.808741 0.585009 0.479873
Tao xong file chua ma tran...
```

- Hàm `mulMatrixFile()`: nhân ma trận rồi chuyển kết quả vào một tập tin khác. Trước hết cần đọc thông tin ma trận: thay vì dùng `scanf()` để nhận dòng và cột từ bàn phím, ta dùng `fscanf()` để nhận chúng từ stream. Các phần tử của ma trận được lưu trữ liên tục trong tập tin như hình ảnh của mảng một chiều. Để tránh cấp phát phức tạp ta dùng mảng một chiều với công thức ánh xạ phần tử `a[i][j]` từ mảng hai chiều `m x n` thành phần tử `b[i*n + n]` trong mảng một chiều, thuận tiện cho việc đọc các phần tử từ tập tin lưu trữ ra mảng một chiều bằng `fscanf()`.

Nhân ma trận có phức tạp một chút vì dùng công thức ánh xạ phần tử. Cuối cùng, việc ghi ma trận kết quả vào tập tin chẳng khác gì xuất chúng ra màn hình, với sự hỗ trợ của các hàm `fputc()` (thay `putchar()`) và `fprintf()` (thay `printf()`).

Cũng giống như hàm `createMatrixFile()`, ta có thể gọi hàm trên như sau:

`createMatrixFile( stdin, stdin, stdout );`

```
2 3 ↵                              dòng và cột ma trận 1
3 2 ↵                              dòng và cột ma trận 2
1.1234 2.4321 3.2135 ↵             các phần tử ma trận 1
4.6352 5.1027 6.0248 ↵
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
7.2413 8.6721 ↵                    các phần tử ma trận 2
9.3425 6.7236 ↵
5.4321 4.3214 ↵
2 2                                dòng và cột ma trận tích
24.487344 22.953648                các phần tử ma trận tích
67.873387 67.915295
Nhan ma tran da xong...
```

Bài 186: (trang 54)

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

typedef struct {
  char name[80];
  long pos;
} INDEX;

long search( FILE* indexf, char* target ) {
  INDEX t;
  rewind( indexf );
  while ( fread( &t, sizeof( INDEX ), 1, indexf ) != 0 )
    if ( strcmp( strlwr( target ), strlwr( t.name ) ) == 0 )
      return t.pos;
  return -1L;
}

void lookIndex( FILE* textf, FILE* indexf ) {
  char line[32];
  printf( "Nhan Ctrl+Z de dung\n" );
  long pos = 0;
  while ( printf( "Ten sach? " ), fgets( line, sizeof( line ), stdin ) ) {
    if ( ( pos = search( indexf, line ) ) == -1 )
      printf( "Khong tim thay sach [%s]", line );
    else if ( fseek( textf, pos, 0 ) != 0 )
      printf( "Vi tri ghi sai%ld\n", pos );
    else
      while( fgets( line, sizeof( line ), textf ) && line[0] != '*' )
        fputs( line, stdout );
  }
}

void createIndex( FILE* textf, FILE* indexf ) {
  INDEX t;
  char line[32];
  while ( ( fgets( line, sizeof( line ), textf ) ) != NULL ) {
    long pos = 0;
    t.pos = pos;
    strcpy( t.name, line );
    while ( fgets( line, sizeof( line ), textf ) && line[0] != '*' ) {}
    /* thông tin kết thúc với dòng bắt đầu bằng dấu * mới được lưu thành chỉ mục */
    if ( line[0] == '*' )
      fwrite( &t, sizeof( INDEX ), 1, indexf );
    pos = ftell( textf );
  }
  printf( "Tao xong file index...\n" );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
}

int main( int argc, char *argv[] ) {
  FILE *textf, *indexf;

  if ( argc != 4 || ( strcmp( argv[1], "-v" ) && strcmp( argv[1], "-i" ) ) ) {
    printf( "Cu phap: LOOK <option> text_file index_file\n"
      "Options: -i  tao index_file\n"
      "         -v  dung index_file tim trong text_file\n" );
    return 0;
  }
  if ( strcmp( argv[1], "-i" ) == 0 ) {
    textf = fopen( argv[2], "r" );
    indexf = fopen( argv[3], "wb" );
    if ( !textf || !indexf ) {
      if ( textf != NULL ) fclose( textf );
      if ( indexf != NULL ) fclose( indexf );
      perror( "Loi" );
      return 0;
    }
    createIndex( textf, indexf );
  } else {
    textf = fopen( argv[2], "r" );
    indexf = fopen( argv[3], "rb" );
    if ( !textf || !indexf ) {
      if ( textf != NULL ) fclose( textf );
      if ( indexf != NULL ) fclose( indexf );
      perror( "Loi" );
      return 0;
    }
    lookIndex( textf, indexf );
  }
  fclose( indexf );
  fclose( textf );
  return 0;
}
```

Với tập tin lớn, chứa nhiều loại dữ liệu khác nhau (ảnh bitmap, ảnh vector 2D hoặc 3D, âm thanh, font, text, …), người ta dùng một tập tin chỉ mục (index) để truy xuất nhanh dữ liệu cần thiết trong tập tin lớn trên. Tập tin chỉ mục này là một tập tin nhị phân chứa các structure (còn gọi là các record - bộ dữ liệu, mẩu tin), mỗi structure lưu các *thông tin lưu trữ* của một dữ liệu tương ứng trong tập tin lớn: loại dữ liệu, offset (vị trí kể từ đầu tập tin lớn), kích thước dữ liệu, … Bài tập trên minh họa cách truy xuất này.

Cần thực hiện phối hợp hai loại thao tác truy xuất tập tin trong bài tập trên:

- Thao tác trên tập tin chỉ mục nhị phân, xem bài tập 179 (trang 260), dùng các hàm: `fwrite()` (khi tạo chỉ mục), `fread()` (khi tìm kiếm, đọc thông tin một chỉ mục).
- Thao tác trên tập tin dữ liệu văn bản, xem bài tập 180 (trang 263), dùng các hàm: `fseek()` (dịch chuyển đến offset dữ liệu cần truy xuất), `fgets()` (đọc từng dòng dữ liệu văn bản).

Xem xét cẩn thận biểu thức điều kiện kiểm tra các tham số dòng lệnh, tuy đơn giản nhưng dễ nhầm lẫn. Chú ý thứ tự các biểu thức điều kiện con:

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Báo lỗi nếu số tham số dòng lệnh khác 4 (`argc != 4`); trong trường hợp số tham số dòng lệnh bằng 4 (nghĩa là `argc != 4` sai) thì báo lỗi nếu tham số thứ nhất khác `"-v"` lẫn (điều kiện `AND`, không phải `OR`) `"-i"`.

Bài 187: (trang 55)

```
#include <stdio.h>
#define MAXCOLS 80
#define MAXROWS 24

int main() {
  char fname[30];
  FILE* fin;
  int nlines, ncols;
  do {
    printf( "Nhap ten file: " );
    scanf( "%29s", fname );
    fin = fopen( fname, "r" );
    if ( !fin ) perror( "Loi" );
  } while ( !fin );
  while ( getchar() != '\n' ) { }
  nlines = ncols = 0;
  while ( !feof( fin ) ) {
    char c = fgetc( fin );
    if ( !feof( fin ) ) {
      ncols++;
      if ( ncols == MAXCOLS || c == '\n' ) {
        nlines++;
        if ( nlines == MAXROWS ) {
          printf( "\nNhan Enter de xem tiep..." );
          getchar();    /* nhận ký tự Enter xem tiếp */
          nlines = 1;
        }
        putchar( c );
        ncols = 1;
      }
      else putchar( c );
    }
  }
  fclose( fin );
  return 0;
}
```

Tương tự bài 175 (trang 256), từng ký tự của tập tin được đọc bằng hàm `fgetc()` và được xuất ra màn hình bằng hàm `putchar()`, cho đến khi đến cuối tập tin, kiểm tra bằng `feof()`.

Ta cần kiểm soát:

- Số ký tự trên mỗi dòng (số cột): bằng cách đếm số ký tự in ra (`ncols`), không cho vượt số ký tự tối đa đề nghị cho mỗi dòng (`MAXCOLS`).
- Số dòng cho mỗi trang: bằng cách đếm số dòng in ra (`nlines`), số dòng này tăng khi ký tự in ra là `'\n'` hoặc `ncols` đã bằng `MAXCOLS` nên cần chuyển đến in ở dòng mới. Khi `nlines` đã bằng số dòng tối đa đề nghị cho mỗi trang (`MAXROWS`), ta dừng lại và xuất dòng nhắc `"\nNhan Enter de xem tiep..."`.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Chú ý khởi tạo lại `ncols = 1` cho mỗi dòng mới và khởi tạo lại `nlines = 1` cho mỗi trang mới.

Bài 188: (trang 55)

```
#include <stdio.h>

int main( int argc, char* argv[] ) {
  FILE *f1, *f2, *f;
  int n1, n2, n;

  if ( argc != 4 ) {
    printf( "Cu phap: SORT file1 file2 sortfile\n" );
    printf( "file1 va file phai sap xep tang truoc\n" );
   return 0;
  }
  f1 = fopen( argv[1], "r" );
  f2 = fopen( argv[2], "r" );
  f = fopen( argv[3], "w" );
  if ( f1 && f2 && f ) {
    FILE *ft;
    int t1, t2;
    t1 = fscanf( f1, "%d", &n1 );
    t2 = fscanf( f2, "%d", &n2 );
    /* đọc từng ký tự bằng fscanf() cho đến cuối một trong 2 file nguồn */
    while ( t1 != EOF && t2 != EOF ) {
      if ( t1 != 1 || t2 != 1 ) { printf( "Loi doc\n" ); break; }
      /* ký tự nào nhỏ hơn đưa vào n, ký tự còn lại dùng so sánh tiếp */
      if ( n1 < n2 ) {
        n = n1;
        /* sau khi đưa ký tự vào n, đọc tiếp file chứa ký tự đó */
        t1 = fscanf( f1, "%d", &n1 );
      } else {
        n = n2;
        t2 = fscanf( f2, "%d", &n2 );
      }
      /* ghi n vào file đích bằng fprintf() */
      if ( fprintf( f, "%d ", n ) == EOF ) { printf( "Loi ghi\n" ); break; }
    }
    /* stream (file) chưa đọc hết đặt tên là ft */
    if ( t1 == EOF && t2 != EOF ) {
      n = n2;
      ft = f2;
    } else if ( t1 != EOF && t2 == EOF ) {
      n = n1;
      ft = f1;
    } else return 0;
    /* đọc tiếp từng ký tự từ stream ft vào stream đích */
    t1 = 1;
    do {
      if ( fprintf( f, "%d ", n ) == EOF ) { printf( "Loi ghi\n" ); break; }
      if ( t1 != 1 ) { printf( "Loi doc\n" ); break; }
    } while ( ( t1 = fscanf( ft, "%d", &n ) ) != EOF );
  }
  if ( f1 ) fclose( f1 );
  if ( f2 ) fclose( f2 );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  if ( f ) fclose( f );
  printf( "Tron ket thuc...\n" );
  return 0;
}
```

Bạn tham khảo bài 81 (trang 141) về thao tác "trộn" (merge) hai mảng (chứa các phần tử tăng/giảm) thành một mảng (chứa các phần tử tăng/giảm). Thao tác này thực hiện giống nhau với các đối tượng tương tự như "run", tập tin chứa các phần tử tăng/giảm, … Chia làm hai giai đoạn:

- Chọn phần tử thích hợp từ một trong hai tập tin nguồn, ghi vào tập tin đích.
- Khi đã duyệt hết phần tử của một trong hai tập tin nguồn, lần lượt ghi các phần tử trong tập tin còn lại vào tập tin đích.

Xem chú thích chi tiết trong bài giải.

Bài 189: (trang 55)

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

int main( int argc, char* argv[] ) {
  FILE *f;
  char *buf, *p;
  char *fname;
  size_t size, readout;
  int i, pos;

  if ( argc > 3 || argc < 2 ) {
    printf( "SPLIT big_file [size_in_Kb]\n" );
    return 0;
  }
  /* xử lý kích thước truyền từ tham số dòng lệnh */
  size = ( argc == 2 ) ? 128 : ( size_t )atol( argv[2] );
  size <<= 10;
  buf = ( char* ) calloc( size, sizeof( char ) );
  if ( !buf ) {
    buf = ( char* ) calloc( 2048, sizeof( char ) );
    if ( !buf ) { perror( "Loi" ); return 1; }
  }

  f = fopen( argv[1], "rb" );
  if ( !f ) { perror( "Loi" ); free( buf ); return 2; }
  /* xử lý tên file lớn để có tên các file con */
  fname = strdup( argv[1] );
  if ( ( p = strrchr( fname, '.' ) ) != NULL ) {}
  if ( p != NULL ) pos = p - fname;
  else             pos = strlen( fname );
  strncpy( fname + pos, ".xxx", 4 );
  pos++;

  i = 0;
  do {
    snprintf( fname + pos, 4, "%03d", i++ );
    /* đọc từ tập tin lớn vào buf */
    readout = fread( buf, sizeof( char ), size, f );
    /* ghi số byte thật sự đọc được từ buf vào từng tập tin con */
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    FILE* f1 = fopen( fname, "wb" );
    fwrite( buf, readout, sizeof( char ), f1 );
    fclose( f1 );
    printf( "File %s [%lu byte(s)]\n", fname, ( unsigned long )readout );
  } while ( readout >= size );

  printf( "Chia file ket thuc... [%d file(s)]\n", i );
  fclose( f );
  free( buf );
  return 0;
}
```

Các vấn đề quan trọng cần giải quyết trong bài tập trên:

- Kích thước các tập tin con: tham số dòng lệnh thứ hai (kích thước tập tin con) được chuyển từ chuỗi thành số `long` bằng hàm `atol()`, kích thước này dùng để cấp phát vùng nhớ đệm `buf`, làm nơi trung chuyển dữ liệu đọc từ tập tin lớn và ghi vào tập tin con.

Chú ý kích thước này tính bằng `Kb` nên cần nhân với 2^{10} để có kích thước tính bằng byte, tương đương với dịch trái `10` đơn vị.

- Tên các tập tin con: tên tập tin lớn được tách lấy phần trước dấu `"."` (hoặc lấy cả tên nếu không có dấu `"."`), kế tiếp được ghép với chuỗi `".xxx"`, các chuỗi ký tự số lần lượt từ `000` trở đi được đặt ngay sau dấu `"."`, chồng lên chuỗi `"xxx"` để tạo tên tập tin con. Hàm `snprintf()` rất hữu dụng khi tạo các chuỗi với định dạng chỉ định như trên.

- Kích thước của tập tin con *cuối cùng*: tập tin con cuối cùng thường có kích thước nhỏ hơn các tập tin trước do là phần lẻ còn lại của tập tin. Mỗi khi đọc một khối byte từ tập tin lớn đến `buf` bằng hàm `fread()`, ta cần lấy trị trả về `readout` của `fread()`, đây là số byte *thật sự đọc được* so với số byte yêu cầu đọc (`size`, tham số thứ ba của `fread()`). Số byte thật sự đọc được `readout` này được dùng như là số byte yêu cầu ghi từ `buf` vào tập tin con, tham số thứ hai của `fwrite()`; cũng dùng như điều kiện kết thúc vòng lặp tách tập tin.

Bài 190: (trang 55)

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

size_t hexDump( FILE *file ) {
  char data[16];
  size_t i, j, size, count = 0;
  /* Dòng tiêu đề */
  fputs( "          ", stdout );
  for ( i = 0; i < sizeof data; ++i )
    printf( "+%lX ", (long unsigned)i );
  /* Phần nội dung, bên trái hex, bên phải ASCII */
  puts( " Contents" );
  do {
    size = fread( data, 1, sizeof data, file );
    if ( size ) {
      /* Địa chỉ offset (base address) */
      printf( "%08lX ", (long unsigned)count );
      count += size;
      /* "hex dump", size là lượng dữ liệu thật sự đọc được */
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
      for ( i = 0; i < size; ++i )
        printf( "%02X ", data[i] );
      /* cho dòng cuối, khi size < sizeof data */
      for ( ++i; i <= sizeof data; ++i )
        fputs( "   ", stdout );
      putchar( ' ' );
      /* "ASCII dump", dữ liệu tương ứng */
      for ( j = 0; j < size; j++ )
        putchar( isprint( data[j] ) ? data[j] : ' ' );
      putchar( '\n' );
    }
  } while ( size == sizeof data );
  return count;
}

int main( int argc, char* argv[] ) {
  FILE* file;
  if ( argc != 2 ) {
    printf( "Syntax: HEXDUMP <filename>\n" );
    return 0;
  }

  file = fopen( argv[1], "rb" );
  if ( file != NULL ) {
    printf( "%lu bytes\n", ( long unsigned )hexDump( file ) );
    fclose( file );
  } else perror( argv[1] );
  return 0;
}
```

Dữ liệu từ tập tin được đọc từng khối 16 byte, ứng với một hàng xuất, vào vùng đệm `data`. Sau đó, dữ liệu *thật sự đọc được* trong vùng đệm `data` được hiển thị bằng hai cách:

- Bên trái: phần "hex dump", dùng chuỗi định dạng "`%X`" để in số hex.
- Bên phải: phần "ascii dump" dùng hàm `putchar()` để in từng ký tự "in được", kiểm tra điều này bằng hàm `isprint()` của `ctype.h`. Trong ASCII, các ký tự "in được" nằm trong khoảng `0x20` đến `0x7E`.

Thực hiện cho đến khi dữ liệu thật sự đọc được nhỏ hơn kích thước vùng đệm, nghĩa là đã hết dữ liệu đọc.

Bài tập không khó, chỉ cần chú ý trình bày thích hợp theo yêu cầu. Xem chú giải chi tiết trong bài giải.

Bài 191: (trang 56)

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

void printCal( int d, int top ) {
  int i, j, n, k;
  n = top - ( 7 - d ) - 28;
  n = ( n > 0 ) ? n : 0;
  for ( j = 1; j <= n; ++j )
    printf( "%5d", top - n + j );
  for ( ; j < d + 1; ++j )
    printf( "%5c", ' ' );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  k = 1;
  for ( ; j <= 7; ++j )
    printf( "%5d", k++ );
  putchar( '\n' );
  for ( i = 2; i <= 5; ++i, putchar( '\n' ) )
    for ( j = 1; j <= 7; ++j ) {
      if ( k > top ) break;
      printf( "%5d", k++ );
    }
}

int main() {
  char* month[] = { "Gieng", "Hai", "Ba", "Tu", "Nam", "Sau",
          "Bay", "Tam", "Chin", "Muoi", "Muoi Mot", "Chap" };
  int m, y, top;
  struct tm tmstruct;

  printf( "Nhap thang, nam (sau 1900): " );
  scanf( "%d%d", &m, &y );
  tmstruct.tm_sec = 0;
  tmstruct.tm_min = 0;
  tmstruct.tm_hour = 0;
  tmstruct.tm_mday = 1;
  tmstruct.tm_mon = m - 1;
  tmstruct.tm_year = y - 1900;
  mktime( &tmstruct );

  switch ( m ) {
    case 4: case 6: case 9: case 11: top = 30; break;
    case 2: top = 28 + ( ( y % 4 == 0 && y % 100 ) || y % 400 == 0 ); break;
    default: top = 31;
  }
  printf( "Thang %s %d\n", month[m - 1], y );
  printf( "   CN  Hai   Ba   Tu  Nam  Sau  Bay\n" );
  printCal( tmstruct.tm_wday, top );
  return 0;
}
```

`time.h` định nghĩa thời gian theo hai dạng:

- *broken-down time*: thời gian tách thành nhóm, nghĩa là thông tin ngày giờ được chia thành các thành phần riêng và được lưu vào trong một cấu trúc đặc biệt do `time.h` định nghĩa: cấu trúc `tm`:

```
struct tm {
  int tm_sec;         /* giây(0-60; 1 giây "nhuận") */
  int tm_min;         /* phút(0-59) */
  int tm_hour;        /* giờ        (0-23) */
  int tm_mday;        /* ngày(1-31) */
  int tm_mon;         /* tháng      (0-11; tính từ tháng 1) */
  int tm_year;        /* năm (tính từ 1900) */
  int tm_wday;        /* thứ (0-6; tính từ Chúa nhật) */
  int tm_yday;        /* ngày trong năm (0-365) */
  int tm_isdst;       /* daylight saving time (-1, 0, 1) */
};
```

"daylight saving time" chỉ dùng khi bạn chọn múi giờ (time zone) thuộc một số nước (ôn đới) và bật chế độ tự động hiệu chỉnh thời gian theo "daylight saving time". Chế

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

độ này tự động điều chỉnh đồng hồ trỏ giờ muộn hơn so với thời gian chuẩn trong mùa xuân tạo cảm giác trời tối muộn hơn trong mùa hè.
Ta có thể thiết lập thời gian chính xác đến giây cho cấu trúc `tm` rồi chuyển đến hàm `mktime()`, sau khi gọi hàm `tm_wday` và `tm_yday` sẽ được hàm tính toán, nghĩa là không cần thiết lập chúng trước khi gọi hàm. Đây là cách bài tập trên dùng.
Cấu trúc `tm` cũng nhận được khi gọi các hàm `gmtime()`, `localtime()`; thực chất là chuyển đổi từ dạng thời gian thứ hai (*calendar time*) sang.
Khi cần lấy thông tin ngày giờ chi tiết ta dùng dạng *broken-down time*.
- *calendar time*: thời gian thể hiện như một số nguyên có kiểu `time_t`, là số giây trải qua từ giây đầu tiên của năm 1900 đến nay. `time_t` nhận được khi gọi hàm `mktime()`; thực chất là chuyển đổi từ dạng thời gian thứ nhất (*broken-down time*) sang.
Khi cần so sánh thời gian, lấy thời gian hiện tại, ta dùng dạng *calendar time*.

Việc tính dòng đầu tiên của lịch tháng trong bài tập được giải quyết như sau:
Trước hết, từ thứ `d` của ngày đầu tháng (`0-6`, tính từ Chúa Nhật) ta có tính được số ngày *đầu tháng* có trên dòng đầu: `7 - d`.
Với `top` là số ngày tối đa trong tháng, nếu số ngày còn lại (`top - (7 - d)`) không đủ phân bố trên 4 dòng tiếp (chứa được 28 ngày), thì dòng đầu có chứa `n` ngày *cuối tháng* chính là số ngày dôi ra. Ngược lại, dòng đầu không chứa ngày cuối tháng (`n = 0`).

```
n = ( top - ( 7 - d ) > 28 ) ? top - ( 7 - d ) - 28 : 0;
```

hoặc:

```
n = ( top - ( 7 - d ) - 28 > 0 ) ? top - ( 7 - d ) - 28 : 0;
```

viết rõ hơn:

```
n = top - ( 7 - d ) - 28;
n = ( n > 0 ) ? n : 0;
```

Như vậy dòng đầu trình bày như sau: từ `1` đến `n` in `n` ngày cuối tháng, từ `n + 1` đến `d` in khoảng trắng, từ `d + 1` đến cuối bắt đầu in những ngày đầu tháng.
Các dòng còn lại in lần lượt các ngày kế tiếp, không được vượt `top`.

Bài 192: (trang 56)

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

int main() {
  char* week[] = { "Chu Nhat", "Thu Hai", "Thu Ba", "Thu Tu",
                   "Thu Nam", "Thu Sau", "Thu Bay" };
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  char* month[] = { "Gieng", "Hai", "Ba", "Tu", "Nam", "Sau",
          "Bay", "Tam", "Chin", "Muoi", "Muoi Mot", "Chap" };
  time_t now, l_now, gmt_now;
  struct tm *t;
  char s[80];

  time( &now );
  t = localtime( &now );
  snprintf( s, 30 + strlen( week[t->tm_wday] ) + strlen( month[t->tm_mon] ),
          "Hom nay %s, ngay %02d thang %s nam ",
          week[t->tm_wday], t->tm_mday, month[t->tm_mon] );
  strftime( s + strlen( s ), 80, "%Y\nBay gio la %I:%M:%S %p", t );
  l_now = mktime( t );
  gmt_now = mktime( gmtime( &now ) );
  printf( "%s %+04g GMT\n", s, difftime( l_now, gmt_now ) / 3600 );
  return 0;
}
```

Thông thường để lấy thông tin ngày và giờ, ta cần phân tích cấu trúc `tm` một cách thủ công. Tuy nhiên, `time.h` cung cấp hàm `size_t strftime( char *str, size_t maxsize, const char *fmt, const struct tm* time );` cho phép chuyển đổi thông tin này từ con trỏ kiểu `tm` *time* thành một chuỗi ký tự với định dạng chỉ định bởi chuỗi *fmt*. Hàm `strftime()` sẽ lưu chuỗi kết quả vào vùng đệm chỉ bởi con trỏ *str*, có kích thước tối đa là *maxsize*.

Các chuỗi định dạng trong `strftime()` (gọi là đặc tả chuyển đổi - conversion specifier) khá chi tiết, nhất là với C99, đủ để trình bày thông tin ngày giờ cho những yêu cầu phức tạp. Bạn tìm hiểu thêm trong những tài liệu về C.

Hàm `time()` trả về thời gian hiện tại dưới dạng `time_t`, bằng hai cách:

`now = time( NULL );  /* now` kiểu `time_t */`

hoặc:

`time( &now );  /*` sau khi gọi hàm `now` thành *calendar time* hiện tại `*/`

Sau đó *calendar time* này được chuyển cho hàm `localtime()` để trả về cấu trúc `tm`, dạng *broken-down time*, cho phép lấy thông tin thời gian chi tiết.

Chuỗi ngày giờ trong bài tập được thực hiện như sau:

- Với thông tin cần chuyển thành tiếng Việt (thứ, tháng) ta phải dùng thông tin lấy từ các thành viên của cấu trúc `tm`, rồi tra trong các bảng lookup (`week`, `month`).
- Với thông tin truy xuất trực tiếp, chuyển cấu trúc `tm` trên cho hàm `strftime()` để tách thông tin thời gian bằng các specifier. Ví dụ: `%Y` (năm), `%I` (giờ, hệ 0-12), `%M` (phút), `%S` (giây), `%p` (`AM` hoặc `PM` cho hệ 0-12), …
- Thông tin về giờ chuẩn căn cứ theo kinh tuyến Greenwich (GMT - Greenwich Mean Time) được tính bằng cách lấy chênh lệch giữa hai *calendar time*:

  Thời điểm hiện tại ở địa phương `l_now`: lấy bằng `localtime()` rồi dùng `mktime()` chuyển sang *calendar time*.

  Thời điểm hiện tại ở kinh tuyến chuẩn Greenwich `gmt_now`: lấy bằng `gmtime()` rồi dùng `mktime()` chuyển sang *calendar time*.

Hàm `difftime()` tính chênh lệch (tính bằng giây) giữa hai *calendar time* này. Vì GMT chia theo múi giờ nên thời gian chênh lệch được chia cho 3600 (số giây/giờ).

Bài tập: Viết hàm sinh chuỗi `code` dựa trên chuỗi `s` đưa vào.

- Bốn ký tự đầu của `code` là bốn ký tự đầu của `s` viết hoa. Nếu có ký tự `space`, hoặc `s` không đủ 4 ký tự, thay thế các ký tự `space` bằng ký tự `'X'`.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

- Năm ký tự cuối của code là năm ký tự cuối của chuỗi thời gian hiện tại.

```
Code: [ABCX03729]
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <time.h>

char* genCode( char *s ) {
  int i;
  size_t size = strlen( s ) < 5 ? strlen( s ) : 5;
  char* t1 = calloc( size, sizeof( char ) );
  char* t2 = calloc( 5, sizeof( char ) );
  char* code = calloc( 9, sizeof( char ) );
  snprintf( t2, 5, "%-4s", strncpy( t1, s, size ) );
  for ( i = 0; i < 5; ++i )
    t2[i] = ( t2[i] == ' ' ) ? 'X' : toupper( t2[i] );
  snprintf( code, 9, "%s%04d", t2, time( NULL ) % 100000 );
  free( t1 );
  free( t2 );
  return code;
}

int main() {
  char *code;
  printf( "Code: [%s]\n", code = genCode( "abc" ) );
  free( code );
  return 0;
}
```

Bài 193: (trang 56)

```
#include <stdio.h>
typedef void funcFile( char* );

void createFile( char* s ) {
  printf( "Create file [%s]\n\n", s );
}

void readFile( char* s ) {
  printf( "Read file [%s]\n\n", s );
}

void adjustFile( char* s ) {
  printf( "Adjust file [%s]\n\n", s );
}

int main() {
  char c;
  funcFile *p[3] = { createFile, readFile, adjustFile };

  do {
    printf( "\nMENU\n" );
    printf( "----\n"
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
            "1. Them\n"
            "2. Doc\n"
            "3. Sua\n"
            "4. Thoat\n");
    printf( "Chon tac vu: " );
    c = getchar();
    while ( getchar() != '\n' ) { }
    if ( c >= '1' && c <= '3' ) p[c - '0' - 1]( "Hello" );
  } while ( c != '4' );
  printf( "Bye...\n" );
  return 0;
}
```

Khác với con trỏ thông thường dùng chỉ đến dữ liệu, con trỏ hàm (function pointer, thường gọi là functor) là một loại con trỏ dùng chỉ đến code. Bạn xem cách dùng con trỏ hàm trong ví dụ cô đọng và chi tiết sau:

```
#include <stdio.h>
/* định nghĩa một số hàm có đặc điểm giống nhau (tham số, trị trả về, ...) */
int add( int a, int b ) { return a + b; }
int mul( int a, int b ) { return a * b; }
/* khai báo con trỏ hàm có tham số và kiểu trả về giống đặc điểm trên */
int ( *pFunc )( int, int );
/* khai báo và gán một mảng các con trỏ hàm */
int ( *apFunc[] )( int, int ) = { add, mul };
/* khai báo một hàm nhận tham số là một con trỏ hàm */
int caller( int, int, int (*p)( int, int ) );

int main() {
  /* gán cho con trỏ hàm một địa chỉ hàm để nó ủy nhiệm đến khi gọi hàm */
  pFunc = &add;
  /* dereference một con trỏ hàm nghĩa là ủy nhiệm lời gọi đến hàm cần gọi
     đây là hàm add(), vừa được gán địa chỉ ở trên */
  printf( "%d\n", ( *pFunc )( 4, 5 ) );
  /* nên dùng cú pháp sau, linh động và dễ sử dụng hơn, ủy nhiệm đến hàm mul()*/
  pFunc = mul;  /* tên một hàm xem như là một con trỏ hàm! */
  /* dùng con trỏ hàm như một hàm! (ủy nhiệm gọi hàm) */
  printf( "%d\n", pFunc( 4, 5 ) );
  /* dùng một con trỏ hàm trong mảng các con trỏ hàm */
  printf( "%d\n", apFunc[0]( 4, 5 ) );
  printf( "%d\n", ( *( apFunc + 1 ) )( 4, 5 ) );
  /* truyền lời gọi hàm mul() đến hàm caller() như là tham số của caller() */
  printf( "%d\n", caller( 4, 5, mul ) );
  return 0;
}

int caller( int a, int b, int (*p)( int, int ) ) {
  return p( a, b );
}
```

Với bài tập cần giải, ta dùng một mảng các con trỏ hàm, mỗi con trỏ hàm chỉ đến một hàm nhận tham số là một chuỗi và có trị trả về là `void`.

Khi khai báo mảng con trỏ hàm thay vì dùng cú pháp:

```
void( *p[] )( char* ) = { createFile, readFile, adjustFile };
```

ta dùng `typedef` để định nghĩa tên kiểu đơn giản và dễ quản lý hơn:

```
typedef void funcFile( char* );
funcFile* p[3] = { createFile, readFile, adjustFile };
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 194: (trang 56)

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

void tab_func( double( *f )( double ), double a, double b, double step ) {
  if ( a > b ) { double t = a; a = b; b = t; }
  printf( "%10s%10s\n", "x", "f(x)" );
  for ( ; a < b + step; a += step )
    printf( "%10.4lf%10.4lf\n", a, ( double )f( a ) );
}

int main () {
  tab_func( sin, 0.0, M_PI, M_PI / 8.0 );
  return 0;
}
```

Hàm yêu cầu viết trong bài tập trên nhận một tham số là một con trỏ hàm. Xem hàm `caller()` trong ví dụ phần giải bài tập 192 (trang 281). Ta phải:
- Khai báo một con trỏ hàm trong danh sách tham số: chú ý cú pháp khai báo phải đủ danh sách tham số và trị trả về.
- Dùng con trỏ hàm trong thân hàm cần viết: do khả năng *dùng con trỏ hàm như một hàm* (gọi là ủy nhiệm gọi hàm) nên cách dùng con trỏ hàm rất tự nhiên.

Bài 195: (trang 57)

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

double average( int n, ... ) {   /* Hàm có danh sách tham số thay đổi */
  int i;
  double s = 0;
  va_list ap;
  va_start( ap, n );
  for ( i = 1; i <= n; ++i ) s += va_arg( ap, double );
  va_end( ap );
  return n ? s / n : 0;
}

int main() {
  double x = 32.4, y = 24.7, z = 4.5, t = 11.8;

  printf( "x = %.1f   y = %.1f   z = %.1f   t = %.1f\n", x, y, z, t );
  printf( "average( 2, x, y )      : %.3f\n", average( 2, x, y ) );
  printf( "average( 3, x, y, z )   : %.3f\n", average( 3, x, y, z ) );
  printf( "average( 4, x, y, z, t ): %.3f\n", average( 4, x, y, z, t ) );
  return 0;
}
```

Hàm yêu cầu trong bài tập là hàm có danh sách tham số thay đổi (variable argument list), còn gọi là *variadic function*. Cú pháp khai báo hàm như sau:

```
fn_type fn_name ( [ arg_type_1 fixed_arg_1,
                    [ arg_type_2 fixed_arg_2, [etc.] ] ]
                    last_arg_type last_fixed_arg, ... );
```

Ví dụ: hàm `printf()`, hàm `scanf()`, …

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Khi khai báo prototype, ngoài các tham số bắt buộc, các tham số còn lại được thay thế bởi chuỗi "...".
Hàm loại này được thực hiện với sự hỗ trợ của các macro trong `<stdarg.h>` sau:

- `void va_start( va_list argptr, last_fixed_arg );`

Dùng khởi tạo danh sách tham số *argptr* trước khi dùng bởi `va_arg()` và `va_end()`; *last_fixed_arg* là tham số cuối bắt buộc khai báo tên trong prototype.

- `void va_arg( va_list argptr, type );`

Mỗi lần gọi sẽ tách từ danh sách tham số *argptr* ra một tham số với kiểu *type*. Lần gọi tiếp sẽ trả về tham số tiếp.

- `void va_end ( va_list argptr );`

Kết thúc xử lý danh sách tham số *argptr*, không lưu các tham số không sử dụng, khi có một lời gọi `va_start()` khác bắt đầu xử lý lại danh sách tham số.

Bài 196: (trang 57)

```
#include <stdio.h>
#define islower( c ) ( (c) >= 'a' && (c) <= 'z' )
#define toupper( c ) ( islower((c))? (c) + ('A'-'a') : (c) )
#define percent( a, b ) ( (b) ? (double)a / b * 100 : 0 )

int main() {
  printf( "'c' viet hoa la '%c'\n", toupper( 'c' ) );
  printf( "123 la %g%% cua 12345\n", percent( 123, 12345 ) );
  return 0;
}
```

Bài 197: (trang 57)

```
#include <stdio.h>
#define BitOn( d, n )   ( (d) | ( (int)1 << (n) ) )
#define BitOff( d, n )  ( (d) & ~( (int)1 << (n) ) )
#define BitFlip( d, n ) ( (d) ^ ( (int)1 << (n) ) )
#define isBit( d, n )   ( ( (d) & ( (int)1 << (n) ) ) > 0 )

void printBits( int d ) {
  int i, size = sizeof( int ) * CHAR_BIT;
  printf( "%d = ", d );
  for ( i = size - 1; i >= 0; --i ) {
    putchar( isBit( d, i ) + 48 );
    if ( i % CHAR_BIT == 0 ) putchar( ' ' );
  }
  putchar( '\n' );
}

int main() {
  int d = 12345;

  printf( "So goc   : " );  printBits( d );
  printf( "Bat bit 8: " );  printBits( BitOn( d, 8 ) );
  printf( "Xoa bit 5: " );  printBits( BitOff( d, 5 ) );
  printf( "Dao bit 4: " );  printBits( BitFlip( d, 4 ) );
  return 0;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Các macro trong bài tập dùng các toán tử bitwise để thực hiện các thao tác trên bit, xem bài tập 56 (trang 111):

- Macro `BitOn( d, n )`: tạo mask bằng cách dùng toán tử dịch trái `<<`, dịch bit `1` đến đúng vị trí cần *bật* bit trong `d`, sau đó dùng toán tử `OR` bitwise để *bật* bit.

- Macro `BitOff( d, n )`: tạo mask bằng cách dùng toán tử dịch trái `<<`, dịch bit `1` đến đúng vị trí, sau đó dùng toán tử đảo bit `~` đảo tất cả bit của mask để có mask đảo. Trong mask đảo, bit `0` nằm đúng vị trí cần *xóa* bit trong `d`, sau đó dùng toán tử `AND` bitwise để *xóa* bit.

- Macro `BitFlip( d, n )`: tạo mask bằng cách dùng toán tử dịch trái `<<`, dịch bit `1` đến đúng vị trí cần *lật* bit trong `d`, sau đó dùng toán tử `XOR` bitwise để *lật* bit.

- Macro `isBit( d, n )`: định trị một *biểu thức điều kiện* để xác định bit đang xem xét là `0` hay `1`. Trước hết, tạo mask bằng cách dùng toán tử dịch trái `<<`, dịch bit `1` đến đúng vị trí cần kiểm tra trong `d`, sau đó dùng toán tử `AND` bitwise để `AND` giữa `d` và mask. Nếu bit kiểm tra bằng `0` thì kết quả `AND` bằng `0`, ngược lại kết quả `AND` lớn hơn `0`. Ta định trị biểu thức điều kiện để bảo đảm trả về `0` hoặc `1`.

Hàm `printBits()` dùng macro `isBit()` này để in các bit của một số.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 198: (trang 57)

```
#include <stdio.h>
#include <limits.h>
#define isBit( d, n ) ( ( (d) & ( (int)1 << (n) ) ) > 0 )
#define LoByte(x) ((x)&(INT_MAX >> sizeof(int)*CHAR_BIT/2-1))
#define HiByte(x) ((x) >> sizeof(int)*CHAR_BIT/2)

void printBits( int d ) {
  int size = sizeof( int ) * CHAR_BIT;
  int i;
  printf( "%d = ", d );
  for ( i = size - 1; i >= 0; --i ) {
    putchar( isBit( d, i ) + 48 );
    if ( i % CHAR_BIT == 0 ) putchar( ' ' );
  }
  putchar( '\n' );
}

int main() {
  int d = INT_MAX / 10;

  printf( "Platform %d-bits\n", sizeof( int ) * CHAR_BIT );
  printf( "INT_MAX / 10:\n" );  printBits( d );
  printf( "High Byte: " );      printBits( HiByte( d ) );
  printf( "Low Byte : " );      printBits( LoByte( d ) );
  return 0;
}
```

Các hằng số sau được khai báo trong `<limits.h>`:
`CHAR_BIT`: số bit trong `1` byte, ít nhất là `8`.
`INT_MAX`: trị `int` lớn nhất, bằng 2^k `- 1` với `k = sizeof(int)*CHAR_BIT`. Như vậy `INT_MAX` có số bit `sizeof(int)*CHAR_BIT - 1`, các bit đều bằng `1`:
`01111111 11111111 11111111 11111111`
Áp dụng để viết các macro như sau:
- Macro `HiByte( x )`: đơn giản dịch phải số integer `x` phân nửa số bit của chúng để loại đi các bit thuộc các byte thấp.

```
Số gốc:        00001100 11001100 | 11001100 11001100

Dịch phải:     00000000 00000000 | 11001100 11001100
```

- Macro `LoByte( x )`: trước hết tạo mask bằng cách dịch phải `INT_MAX` phân nửa số bit `- 1` (toán tử - ưu tiên hơn toán tử >>), sau đó `AND` bitwise cho `x` để lọc lấy các bit thuộc các byte thấp.

```
Số gốc:        00001100 11001100 | 11001100 11001100
Mask:          00000000 00000000 | 11111111 11111111
Kết quả AND:   00000000 00000000 | 11001100 11001100
```

Bài 199: (trang 57)

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

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
char* firstname( char* s ) {
  char *t = strdup( t );
  char *r = strtok( strchr( t, ':' ), " :" );
  free( t );
  return r;
}

int compare( const void* s, const void* s1 ) {
  char *t, * t1;
  t = firstname( *( char** ) s );
  t1 = firstname( *( char** ) s1 );
  if ( strcmp( t, t1 ) == 0 ) {
    t = strchr( *( char** ) s, ' ' );
    t1 = strchr( *( char** ) s1, ' ' );
  }
  return strcmp( t, t1 );
}

int main() {
  int i, n;
  char* s[100];
  FILE *f;

  f = fopen( "EMP.TXT", "r" );
  if ( !f ) {
    perror( "Loi" );
    return 1;
  }

  printf( "Noi dung...\n" );
  i = 0;
  while ( ( s[i] = calloc ( 255, sizeof( char ) ) ) != NULL
          && fgets( s[i], 255, f ) != NULL )
    printf( "%s", s[i++] );
  fclose( f );

  printf( "Sap xep...\n" );
  n = i;
  qsort( ( void* )s, ( size_t )n, sizeof( char* ), compare );
  for ( i = 0; i < n; ++i )
    printf( "%s", s[i] );
  return 0;
}
```

Giải thuật sắp xếp được xem là nhanh nhất là sắp xếp nhanh (quick sort). Nội dung giải thuật (sắp xếp tăng) như sau:

```
void quicksort( int a[], int lo, int hi ) {
  int i = lo, j = hi, pivot;
  if ( i < j ) {
    pivot = a[hi];
    do {
      while ( i < j && a[i] < pivot ) i++;
      while ( j > i && a[j] >= pivot ) j--;
      if ( i < j ) swap( a[i], a[j] );
    } while ( i < j );
    swap( a[i], a[hi] );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    quicksort( a, lo, i - 1 );
    quicksort( a, i + 1, hi );
  }
}
```

- Chọn một phần tử làm trục `pivot`.
- Mảng cần sắp xếp được phân hoạch (partition) thành hai phần với phần tử `pivot` làm trục sao cho: các phần tử từ vị trí `0` đến vị trí `pivot - 1` có nội dung nhỏ hơn phần tử `pivot`, các phần tử từ vị trí `pivot + 1` đến vị trí `n - 1` có nội dung lớn hơn hoặc bằng phần tử `pivot`.
- Tiếp tục chọn phần tử `pivot` và phân hoạch cho mỗi phần trong hai phần trên ...

Đây là một quá trình đệ quy với điểm dừng là khi danh sách đã có thứ tự.

Về bản chất, quick sort lần lượt chọn các phần tử trong danh sách làm phần tử `pivot`, phần tử này sẽ được sắp xếp đúng vị trí, khi tất cả các phần tử đều được chọn làm phần tử `pivot` thì mảng sắp xếp xong, đệ quy sẽ dừng lại.

Giải thuật phân hoạch với một phần của mảng, giới hạn bởi biên dưới `lo` và biên trên `hi`, được thực hiện như sau:

- Chọn phần tử cuối `a[hi]` làm trục.
- Quét mảng theo 2 hướng, từ `lo` lên (`i++`) và từ `hi` xuống (`j--`), nếu vẫn "bình thường" (sắp xếp đúng) thì tiếp tục quét (2 vòng `while` bên trong), nếu "sai" (sắp xếp sai) thì hoán đổi.
- Sau khi quét xong (thoát khỏi vòng `do while`, phần tử `pivot` đã sắp đúng vị trí), hoán đổi vị trí `a[hi] (pivot)` với vị trí `a[i]`.

Giải thuật quick sort được áp dụng trong hàm `qsort()` của `stdlib.h`:

```
void qsort( void* array, size_t n, size_t size,
            int ( *compare )( const void*, const void* ) );
```

àm `qsort()` sẽ sắp xếp mảng *array* bằng thuật toán quick sort theo điều kiện do người dùng quy định. Người dùng định nghĩa điều kiện sắp xếp bằng cách định nghĩa một hàm callback (hai tham số) so sánh hai phần tử của mảng, các trị trả về của hàm này (`< 0`, `= 0`, `> 0`) quyết định chiều sắp xếp. `qsort()` gọi hàm này thông qua con trỏ hàm được truyền như tham số cuối của `qsort()`.

```
int ( *compare )( const void *, const void * );
```

Tham số thứ hai (kiểu `size_t`) là số phần tử cần sắp xếp.

Tham số thứ ba là kích thước mỗi phần tử của *array*, do *array* được `qsort()` chuyển thành kiểu `void*` nên cần tham số này để xác định kích thức mỗi phần tử.

Chú ý là hàm callback chuyển tham số truyền tới thành con trỏ kiểu `void*`, bên trong hàm ta ép kiểu tham số rồi dùng toán tử lấy nội dung `*` (dereference - giải quy) để so sánh. Ví dụ cần so sánh 2 phần tử kiểu `int`, tham số được truyền như con trỏ *`void*`*, ta ép kiểu thành `int*` (không phải thành `int`), rồi dùng toán tử dereference `*`:

```
int intcompare( const void* a, const void* b ) {
  return *( ( int* ) a ) - *( ( int* ) b );
}
```

Sẽ dễ nhầm lẫn nếu ta cần so sánh 2 chuỗi (nghĩa là hai con trỏ `char*`), lúc đó ta ép kiểu thành con trỏ chỉ đến con trỏ:

```
int strcompare( const void* sp1, const void* sp2 ) {
  const char* s1 = *( char** )sp1;
  const char* s2 = *( char** )sp2;
  return strcmp( s1, s2 );
}
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Hàm `firstname()` dùng `strchr()` để xác định token thứ hai, tách `firstname` từ token này bằng `strtok()` với delimiter là `space` và `":"`. Delimiter `":"` nhằm loại dấu `":"` đầu chuỗi do `strchr()` trả về và đề phòng trường hợp không có `lastname`.

`strtok( strchr( t, ':' ), " :" )`

Vì `strtok()` phá hủy chuỗi phân tích nên ta sao chép ra chuỗi `t` trước khi phân tích. Sau khi so sánh thấy `firstname` giống nhau, ta lại dùng `strchr()` tìm ký tự `space` phân cách giữa `firstname` và `lastname` để xác định `lastname`.

Bài 200: (trang 58)

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

char* firstname( char* s ) {
  char *t = strdup( t );
  char *r = strtok( strchr( t, ':' ), " :" );
  free( t );
  return r;
}

int compare( const void* s, const void* s1 ) {
  char *t, * t1;
  t = firstname( *( char** ) s );
  t1 = firstname( *( char** ) s1 );
  return strcmp( t, t1 );
}

int compare1( const void* s, const void* s1 ) {
  return strcmp( *( char** ) s, *( char** ) s1 );
}

int main() {
  int i, n;
  char *s[100], *fname[100], *t;
  char** p;
  char key[20];
  char* k = key;
  FILE *f;

  f = fopen( "PERSON.TXT", "r" );
  if ( !f ) {
    perror( "Loi" );
    return 1;
  }
  printf( "Noi dung...\n" );
  i = 0;
  while ( ( s[i] = calloc ( 255, sizeof( char ) ) ) != NULL
    && ( fname[i] = calloc ( 20, sizeof( char ) ) ) != NULL
    && fgets( s[i], 255, f ) != NULL ) {
    printf( "%s", s[i] );
    fname[i] = firstname( s[i] );
    i++;
  }
  fclose( f );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  n = i;
  qsort( ( void* )s, ( size_t )n, sizeof( char* ), compare );
  qsort( ( void* )fname, ( size_t )n, sizeof( char* ),  compare1 );

  printf( "\nNhap firstname: " );
  fgets( key, 20, stdin );
  if ( ( t = strrchr( key, '\n' ) ) != NULL ) *t = '\0';

  p = ( char** )bsearch( ( void * ) &k, ( void * ) fname,
      ( size_t )n, sizeof( char * ), compare1 );
  if ( p )
    printf( "Ma vung cua %s: %s\n",
          key, strtok( strchr( s[p-fname], '(' ), "()" ) );
  else
    printf( "Khong tim thay %s\n", key );
  return 0;
}
```

Với mảng *đã sắp xếp*, người ta thường dùng thuật toán tìm kiếm nhị phân (binary search) để xác định nhanh một phần tử trong mảng.
Hàm `bsearch()` của `stdlib.h` dùng thuật toán tìm kiếm nhị phân để tìm một phần tử so trùng với *`key`* trong một mảng *đã sắp xếp array*:

```
void bsearch( const void* key, const void* array, size_t n,
            size_t size, int ( *compare )( const void*, const void* ) );
```

Ba tham số cuối của `bsearch()` giống hàm `qsort()`, xem bài 199 (trang 288), đặc biệt là tham số cuối là con trỏ hàm chỉ đến hàm so sánh do người dùng viết, để cho biết mảng *array* sắp xếp theo điều kiện nào.

Trước khi chuyển sang phần Cấu trúc dữ liệu, chúng ta ôn tập bằng cách thực hiện vài bài tập có đặc tả phức tạp. Do bài giải dài, nên chúng tôi chỉ kèm theo source code của sách, không trình bày ở đây.

Bài tập:
Chương trình tính thuế thu nhập (Ontario, Canada) dựa trên tài liệu đặc tả sau:
Payroll Deduction Calculations for Ontario Employees

I Gross Income
F Registered Pension Plan Contributions
U1 Union Dues
P Number of Pay Periods
TC Federal Personal Claim
TCP Provincial Personal Claim
XX Number of Dependants below age of 18
YY Labour-sponsored share purchases
F1 Authourized deductions
D CPP contributions year-to-date
C CPP contribution: trị nhỏ hơn giữa
(4.95% * (I - $3,500 / P), < 0 thì dùng 0) và (1,861.20 – D, ≥ 0)
D' EI premiums year-to-date
EI EI premiums: trị nhỏ hơn giữa (1.95% * I) và ($760.50 – D', ≥ 0)
A Annual Taxable Income : P * (I - F - U1) - F1

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

1. Annual Federal tax (**T1**)

Annual Federal Tax Bracket (**FF**)

Income Bracket	1	≤ 35,595	> 35,595 ≤ 71,190	> 71,190 ≤ 115,739	> 115,739
Annual Taxable Income **A**	2				
Rate **R**	3	16%	22%	26%	29%
A * R	4				
Overcharge Constant **K**	5	0	2,136	4,983	8,455
A * R - K	**FF**				

K1 Personal Tax Credit: 16% * TC

K2 CPP, EI credit: 16% * (P * C, tối đa 1,861.20) + 16% * (P * EI, tối đa 760.50)

T3 Basic Federal Tax: FF - K1 - K2

LCF Labour-sponsored tax credit: trị nhỏ hơn giữa $750 và 15% * YY

T1 = T3 - LCF

2. Annual Ontario Tax (**T2**)

Annual Ontario Tax Bracket (**PP**)

Income Bracket	6	≤ 34,010	> 34,010 ≤ 68,020	> 68,020
Annual Taxable Income **A**	7			
Rate **V**	8	6.05%	9.15%	11,16%
A * V	9			
Overcharge Constant **KP**	10	0	1,054	2,422
A * V - KP	**PP**			

K1P Personal Tax Credit: 6.05% * TCP

K2P CPP, EI credit: 6.05% * (P * C, tối đa 1,861.20) + 6.05% * (P * EI, tối đa 760.50)

T4 Basic Provincial Tax: PP - K1P - K2P

V1 Total Surtax

Surtax a: (T4 - 3,929) * 20%, ≥ 0

Surtax b: (T4 – 4,957) * 36%, ≥ 0

S Provincial Tax Reduction: nhỏ hơn giữa

(T4 + V1) và (2 * ($190 + $350 * XX) - (T4 + V1), < 0 thì dùng 0)

LCP Labour-sponsored tax credit: nhỏ hơn giữa $750 và 15% * YY

Ontario Health Premium (**V2**)

Income Bracket	11	≤20,000	>20,000 ≤36,000	>36,000 ≤48,000	>48,000 ≤72,000	>72,000 ≤200,000	>200,000
Annual Taxable Income **A**	12						
Base Amount	13	0	20,000	36,000	48,000	72,000	200,000
(12) - (13)	14						
Rate	15	0%	6%	6%	25%	25%	25%
(14) * (15)	16						
Base Premium	17	0	0	300	450	600	750
(16) + (17)	18						
Premium Limit	19	0	300	450	600	750	950
Min của (18) (19)	**V2**						

T2 = T4 + V1 + V2 - S - LCP

TP1 Annual Federal Tax per Pay Period: T1 / P

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

TP2 Annual Ontario Tax per Pay Period: T2 / P
T Total Tax Deduction per Pay Period: TP1 + TP2
TD Total Deductions: T + C + EI + F + U1
Net Income: I - TD

```
Payroll Deduction Calculator
============================

Please enter the employee's pay period information :
 Income for the current pay period     : 1600
 Registered Pension Plan contributions : 57
 Union dues for the current pay period : 7
Please enter the employee's annual information :
 Number of pay periods this year       : 26
 Federal tax credits (as per TD1 form) : 8148
 Ontario tax credits (as per TD1 form) : 8196
 Number of Dependants < 18 years old   : 2
 Labor-sponsored share purchases       : 0
 Authorized deductions                 : 0
Please enter the employee's year-to-date information :
 CPP contributions year-to-date        : 500
 EI premiums paid year-to-date         : 200

Gross Income                1600.00
Deductions:
  Federal Tax                189.49
  Provincial Tax              92.14
  Canada Pension Plan         72.54
  Employment Insurance        31.20
  RRSP Contributions          57.00
  Union Dues                   7.00
  Total Deductions           449.37
Net Income                  1150.63
```

Bài tập:
Tài khoản (account) gồm nhiều loại. Tại một thời điểm, tài khoản có một số dư (balance). Ghi nợ (debit) và ghi có (credit) là hai loại bút toán thực hiện trên tài khoản. Tác động của nó khác nhau tùy theo loại tài khoản:
- Tài khoản có số dư nợ (debit balances): ghi nợ (debit) làm tăng balance của nó, ghi có (credit) làm giảm balance của nó. Tài khoản có số dư nợ gồm các loại sau:
  + Asset account: tài khoản tài sản (tiền mặt, bất động sản, thiết bị, hàng tồn kho, khoản phải thu). Mã bắt đầu 1 (tài sản ngắn hạn), 2 (tài sản dài hạn).
  + Expense account: tài khoản chi phí (chi phí hoạt động, khấu hao, lãi vay phả trả). Mã bắt đầu 6 (chi phí).
- Tài khoản có số dư có (credit balances): ghi nợ (debit) làm giảm balance của nó, ghi có (credit) làm tăng balance của nó. Tài khoản có số dư có gồm các loại sau:
  + Liability account: tài khoản nợ phải trả (các khoản vay, nợ dài hạn, lãi phải trả). Mã bắt đầu 3 (nợ phải trả).
  + Equity account: tài khoản đầu tư (khoản dự phòng, lợi nhuận giữ lại). Mã bắt đầu 4 (nguồn vốn chủ sở hữu).
  + Revenue account: tài khoản doanh thu (doanh thu dịch vụ, tiền lãi nhận được). Mã bắt đầu 5 (doanh thu).

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Thực tế có nhiều loại tài khoản hơn. Mã tài khoản có từ 3 đến 4 ký tự, tùy theo mức độ chi tiết.

Giao tác (transaction) gọi là hạch toán, bao gồm một số bút toán thực hiện trên vài tài khoản có liên quan.

Ví dụ: một số hạch toán.

- Mua laptop trả bằng tiền mặt:
  Nợ tài khoản 2112 (Equipment & machine, thiết bị & máy móc) giá trị máy nhập về.
  Có tài khoản 111 (Cash on hand, tiền mặt) tiền chi mua máy.
- Doanh thu cung cấp dịch vụ với thuế giá trị gia tăng (VAT) 5%:
  Nợ tài khoản 111 tổng giá trị thanh toán.
  Có tài khoản 511 (Sales, tài khoản doanh thu bán hàng) doanh thu chưa có VAT
  Có tài khoản 3331 (Value Added Tax, tài khoản thuế giá trị gia tăng) VAT phải nộp.
- Chi phí điện nước mua ngoài:
  Nợ tài khoản 6277 (Outside purchasing services cost, chi phí dịch vụ mua ngoài).
  Có tài khoản 111 chi

Thực tế, nhiều hạch toán rất phức tạp, nhưng chúng đều bảo đảm dư nợ và dư có của tất cả các tài khoản phải cân bằng nhau. Mất cân đối xảy ra thường do bút toán sai.

Nhật ký kế toán ghi nhận các giao tác trên:

```
General Journal
Account Description                          Debit     Credit
------- ------------------------------ ---------- ----------
    411 Von kinh doanh                                 100000
    111 Tien mat tu von                     100000
    111 Chi tien mua laptop                                3500
   2112 Nhap tai san: laptop (3500)           3500
    111 Thu tien dich vu                      12600
    511 Doanh thu dich vu #1234                         12000
   3331 VAT của #1234                                     600
    111 Chi tien hoa don dien nuoc                        5000
    641 Chi phi dien nuoc                      5000
    111 Agribank chuyen tien mat              22000
    311 Vay Agribank                                    22000
```

Sổ kế toán cân đối từng loại tài khoản và cho biết dư nợ, dư có của loại tài khoản đó.

```
Balance Sheet
Account Description                          Debit     Credit
------- ------------------------------ ---------- ----------
    111 Cash on hand                        134600       8500
   2112 Equipment & machine                   3500          0
    641 Selling expenses                      5000          0
            ------------------------------------------------
            Debit Balances                  134600
-------------------------------------------------------------
    411 Working capital                          0     100000
    311 Short-term loan                          0      22000
   3331 Value Added Tax                          0        600
    511 Sales                                    0      12000
            ------------------------------------------------
            Credit Balances                                 134600
-------------------------------------------------------------
```

Báo cáo thu nhập cho thấy thu nhập ròng (net income) là hiệu của tổng doanh thu (revenues) với tổng chi phí (expenses):

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
Income Statement
Account Description                          Debit     Credit
------- ------------------------------ ---------- ----------
    511 Sales                                   0      12000
                                       ---------- ----------
            Total Revenues                              12000
-------------------------------------------------------------
    641 Selling expenses                     5000          0
                                       ---------- ----------
            Total Expenses                   5000
-------------------------------------------------------------
            Net Income                                7000.00
-------------------------------------------------------------
```

Người sử dụng chương trình dùng hai menu:
Menu thứ nhất hiển thị các tác vụ cơ bản: thực hiện giao tác, xem nhật ký kế toán, xem sổ kế toán, xem báo cáo thu nhập.

```
------------ MENU ------------
1. Execute transaction
2. View the general journal
3. View the balance sheet
4. View the income statement
5. Quit the program
------------------------------
```

Khi chọn thực hiện giao tác (mục 1.), menu thứ hai sẽ hiển thị loại giao tác: tạo tài khoản, ghi nợ và ghi có đến tài khoản chỉ định.

```
------ Transaction MENU ------
1. Create account
2. Debit
3. Credit
4. Cancel
------------------------------
```

Bài tập trên chỉ khó ở chỗ bạn phải phân tích được đặc tả với kiến thức thuộc về một ngành khác. Sau khi phân tích, bài tập được thực hiện dễ dàng khi tổ chức tốt cấu trúc dữ liệu dùng chứa tài khoản (account) và giao tác (transaction).

Bài 201: (trang 59)

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

struct NODE {
  int data;
  struct NODE* next;
};
typedef struct NODE* NODEPTR;

void InsertFirst( NODEPTR* l, int x ) {
  NODEPTR p = ( NODEPTR )malloc( sizeof( struct NODE ) );
  p->data = x;
  p->next = *l;
  *l = p;
}

void InsertLast( NODEPTR* l, int x ) {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  NODEPTR p = ( NODEPTR )malloc( sizeof( struct NODE ) );
  p->data = x;
  p->next = NULL;
  if ( !*l ) *l = p;
  else {
    NODEPTR q;
    for ( q = *l; q->next; q = q->next ) { }
    q->next = p;
  }
}

void OutList( NODEPTR l, char* s ) {
  printf( "%s: ", s );
  for ( ; l ; l = l->next ) printf( "[%d]", l->data );
  putchar( '\n' );
}

void AfterInsert( NODEPTR l, int d, int x ) {
  if ( !l ) return;
  if ( l->data == d ) {
    NODEPTR p = ( NODEPTR )malloc( sizeof( struct NODE ) );
    p->data = x;
    p->next = l->next;
    l->next = p;
    return;
  }
  AfterInsert( l->next, d, x );
}

int main() {
  NODEPTR l = NULL;
  int x;

  printf( "Nhap 0 de dung: " );
  do {
    scanf( "%d", &x );
    if ( x ) InsertFirst( &l, x );
  } while ( x );
  OutList( l, "List goc" );

  InsertLast( &l, 5 );
  OutList( l, "Chen 5 cuoi" );
  AfterInsert( l, 3, 4 );
  OutList( l, "Chen 4 sau node [3]" );
  return 0;
}
```

Chèn node vào danh sách liên kết được viết như một thành phần của hàm nhập để đưa dữ liệu vào danh sách liên kết.

1. Chèn node đầu

Chuỗi dữ liệu được chèn đầu vào danh sách liên kết sẽ có thứ tự lưu trữ ngược với thứ tự nhập (nghĩa là cũng ngược với thứ tự duyệt - LIFO). Các bước:

- Cấp phát một node `p` mới và đưa đầy đủ dữ liệu vào node.
- Cho `p->next` và `l` trỏ cùng một nơi (chứa cùng địa chỉ): `p->next = l`.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

- l bây giờ thay đổi thành p: l = p.

```
void Solution( NODEPTR* l, int x ) {
  NODEPTR p = ( NODEPTR )malloc( sizeof( struct NODE ) );
  p->data = x;
  p->next = *l;
  *l = p;
}
```

Cần nắm vững hai bước cuối.
Khi viết hàm nhập, sử dụng thao tác chèn này, con trỏ l cung cấp cho hàm nhập *phải khởi tạo trước* bằng NULL trước.

2. Chèn sau node q
Theo quan điểm node bất kỳ của danh sách cũng là node đầu (node đầu của danh sách còn lại), ta thấy thực chất việc chèn sau node p là chèn node đầu, với con trỏ quản lý danh sách l bây giờ là q->next (xem lại chèn node đầu):
Phải kiểm tra xem l và q có khác NULL không trước khi chèn.

```
void Solution( NOPDEPTR l, NOPDEPTR q, int x ) {
  if ( l && q ) {
    NODEPTR p = ( NODEPTR )malloc( sizeof( struct NODE ) );
    p->data = x;
    p->next = q->next;
    q->next = p;
  }
}
```

Trong trường hợp chèn sau một node chứa trị d, dùng đệ quy:
- Đệ quy cho đến khi tìm ra node q chứa trị d. Khi tìm ra node q thực hiện chèn (chèn đầu với l bây giờ là l->next) rồi kết thúc đệ quy.
- Có hai trường hợp dừng đệ quy: đã đến cuối danh sách hoặc đã tìm và chèn được node.

```
void Solution( NODEPTR l, int d, int x ) {
  if ( !l ) return;
  if ( l->data == d ) {
    NODEPTR p = ( NODEPTR )malloc( sizeof( struct NODE ) );
    p->data = x;
    p->next = l->next;
    l->next = p;
    return;
  }
  Solution( l->next, d, x );
}
```

3. Chèn node cuối
Chuỗi dữ liệu được chèn đầu vào danh sách liên kết sẽ có thứ tự lưu trữ giống với thứ tự nhập (nghĩa là cũng giống với thứ tự duyệt - FIFO). Vì vậy cách chèn cuối thường được lựa chọn hơn.
- Cấp phát một node p mới và đưa đầy đủ dữ liệu vào node (với p->next = NULL).

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

- Nếu danh sách rỗng thì node mới là node đầu tiên nên `l = p`.
- Nếu không, tìm đến node cuối `q` (không phải tìm con trỏ cuối). Ta dùng một vòng lặp đưa node `q` chạy từ đầu danh sách đến cuối, nghĩa là khi con trỏ cuối của danh sách là `q->next` có trị `NULL`. Đây chính là thao tác "dò phía trước".
- Con trỏ cuối này phải chỉ đến `p`: `q->next = p`.

```
void Solution( NODEPTR* l, int x ) {
  NODEPTR p = ( NODEPTR )malloc( sizeof( struct NODE ) );
  p->data = x;
  p->next = NULL;
  if ( !*l ) *l = p;
  else {
    NODEPTR q;
    for ( q = *l; q->next; q = q->next ) { }
    q->next = p;
  }
}
```

Khó khăn do việc phải duyệt đến cuối danh sách liên kết có thể được khắc phục bằng cách đưa thêm con trỏ cuối (thường gọi là con trỏ `last`) vào cấu trúc và chấp nhận thêm một số thao tác để xử lý con trỏ cuối này.
Với số lượng node ít, chèn node cuối có thể thực hiện bằng đệ quy:
- Đệ quy với phần còn lại của danh sách *cho đến con trỏ cuối*.
- Khởi tạo con trỏ cuối này thành một node và đưa đầy đủ dữ liệu vào node.

```
void Solution( NODEPTR* l, int x ) {
  if ( !*l ) {
    *l = ( NODEPTR )malloc( sizeof( struct NODE ) );
    ( *l )->data = x;
    ( *l )->next = NULL;
    return;
  }
  Solution( &( *l )->next, x );
}
```

4. Nhập đệ quy

Cách nhập đệ quy cho kết quả giống *chèn node cuối*, thuận tiện do chuỗi lưu trữ có thứ tự giống với chuỗi nhập. Cách này cũng không dùng hàm chèn node, không cần khởi tạo `NULL` cho con trỏ đầu danh sách. Vì vậy thích hợp cho việc viết nhanh các ví dụ. Khuyết điểm cách nhập này là chỉ áp dụng với danh sách liên kết có số node giới hạn để hạn chế bậc đệ quy.
Ví dụ sau sẽ nhập các số nguyên vào danh sách liên kết cho đến khi nhập số `0`.

```
NODEPTR Solution() {
  int x;
  NODEPTR p;
  scanf( "%d", &x );
  if ( !x ) return NULL;
  p = ( NODEPTR )malloc( sizeof( struct NODE ) );
  p->data = x;
  p->next = Solution();
  return p;
}

int main() {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  NODE l;
  printf( "Nhap 0 de dung: " );
  l = Solution();
  /* ... */
}
```

Bài 202: (trang 59)

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

struct NODE {
  int data;
  struct NODE* next;
};
typedef struct NODE* NODEPTR;

NODEPTR InList() {
  int x;
  NODEPTR p;
  scanf( "%d", &x );
  if ( !x ) return NULL;
  p = ( NODEPTR )malloc( sizeof( struct NODE ) );
  p->data = x;
  p->next = InList();
  return p;
}

void Solution( NODEPTR l ) {
  if ( !l ) return;
  if ( l->data % 2 == 0 ) {        /* điều kiện là trị chẵn */
    NODEPTR p = ( NODEPTR )malloc( sizeof( struct NODE ) );
    p->data = 0;
    p->next = l->next;
    l->next = p;
    Solution( l->next->next );     /* vượt qua node vừa chèn */
  }
  else Solution( l->next );
}

void OutList( NODEPTR l, char* s ) {
  printf( "%s: ", s );
  for ( ; l ; l = l->next ) printf( "[%d]", l->data );
  putchar( '\n' );
}

int main() {
  NODEPTR l;

  printf( "Nhap 0 de dung: " );
  l = InList();
  OutList( l, "List goc" );

  Solution( l );
  OutList( l, "Chen 0 sau tri chan" );
  return 0;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
}
```

Ta dùng cách chèn đệ quy, nhưng chú ý: trong trường hợp phải chèn sau *tất cả các* node chứa trị `d` với điều kiện nào đó, khi chèn một node xong sẽ *không dừng đệ quy* mà phải "vượt" qua node vừa chèn vào. Vì nếu node chèn vào có cùng trị `d` (hoặc có điều kiện ràng buộc như `d`) sẽ gây đệ quy vĩnh viễn làm tràn stack.

Bài 203: (trang 60)

```
void Solution( NODEPTR* l, int x ) {
  NODEPTR p = ( NODEPTR )malloc( sizeof( struct NODE ) );
  p->data = x;
  if ( !*l || ( *l )->data > x ) {
    p->next = *l;
    *l = p;
  } else {
    NODEPTR t;
    for ( t = *l; t->next && t->next->data < x; t = t->next ) { }
    p->next = t->next;
    t->next = p;
  }
}

/* gọi trong hàm InList() */
NODEPTR InList() {
  int x;
  NODEPTR p = NULL;
  while ( 1 ) {
    scanf( "%d", &x );
    if ( !x ) return p;
    Solution( &p, x );
  }
}
```

Giải pháp không đệ quy như sau:

- Nếu danh sách rỗng hoặc node đầu chứa trị lớn hơn trị cần chèn, ta thực hiện thao tác *chèn đầu*.
- Nếu không, dùng phương pháp "dò phía trước", xác định *node trước* node cần chèn và thực hiện thao tác *chèn sau* một node bất kỳ.

Một cách trình bày khác, dễ hiểu hơn, cho giải pháp trên:

```
void Solution( NODEPTR* l, int x ) {
  NODEPTR t1, t2;
  NODEPTR p = ( NODEPTR )malloc( sizeof( struct NODE ) );
  p->data = x;
  p->next = NULL;
  t1 = NULL;
  t2 = *l;
  while ( t2 && t2->data < x ) {
    t1 = t2, t2 = t2->next;
  }
  if ( !t1 ) { p->next = *l; *l = p; }
  else       { p->next = t1->next; t1->next = p; }
}
```

- Đầu tiên tạo một node `p` kiểu `NODEPTR` chứa trị `x`.
- Tìm node `x` trong danh sách `l` bằng con trỏ `t2`, con trỏ `t1` đi ngay sau `t2`. Đây là kỹ

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

thuật "hai con trỏ theo nhau": trước khi `t2` dịch chuyển, gán `t1` bằng `t2` để lưu node trước (hoặc node cha) của `t2`. Khi vòng lặp dừng, `t1` sẽ là node trước vị trí cần chèn. Nếu `t1` bằng `NULL`, nghĩa là vòng lặp không hoạt động do danh sách `l` rỗng hoặc node đầu đã có dữ liệu lớn hơn `x`, ta *chèn đầu* node `p` vào danh sách `l`.
Ngược lại, chèn node `p` vào danh sách `l` theo giải thuật *chèn sau* node `t1`.
Giải pháp đệ quy rõ ràng ngắn gọn hơn:
- Đệ quy cho đến khi gặp một trong hai trường hợp dừng đệ quy: xác định được node có trị lớn hơn trị cần chèn hoặc đã đến cuối danh sách.
- Con trỏ `l` lúc này được xem như một con trỏ "chỉ đến node đầu" của "danh sách còn lại", ta thực hiện thao tác chèn đầu rồi chấm dứt đệ quy.

```
void Solution( NODEPTR* l, int x ) {
  if ( !*l || ( *l )->data > x ) {
    NODEPTR p = ( NODEPTR )malloc( sizeof( struct NODE ) );
    p->data = x;
    p->next = *l;
    *l = p;
    return;
  }
  Solution( &( *l )->next, x );
}
```

Một giải pháp cũng đáng quan tâm là chèn phía trước danh sách liên kết một *node giả* (ghost node), như vậy loại bỏ được việc xét danh sách rỗng và trường hợp chèn đầu. Tuy nhiên thao tác chuẩn bị phức tạp, chỉ sử dụng nếu node giả có thể sử dụng nhiều lần.

```
#include <stdio.h>
void Solution( NODEPTR l, int x ) {
  NODEPTR p = ( NODEPTR )malloc( sizeof( struct NODE ) );
  p->data = x;
  for ( ; l->next && l->next->data < x; l = l->next ) { }
  p->next = l->next;
  l->next = p;
}

int main() {
  /* ... */
  /* ghost node */
  q = ( NODEPTR )malloc( sizeof( struct NODE ) );
  q->next = l;        /* chèn đầu node giả vào danh sách l */
  Solution( q, x );   /* chèn vào danh sách khởi đầu từ node giả q */
  l = q->next;        /* bắt buộc phải dùng do có thể có chèn đầu */
  /* ... */
  return 0;
}
```

Bài 204: (trang 60)
Thông thường node cần xóa do một hàm tìm node theo một yêu cầu nào đó trả về. Ví dụ xóa node có trị bằng `x`, hàm tìm node sẽ là:

```
NODEPTR Find( NODEPTR l, int x ) {
  for ( ; l; l = l->next )
    if ( l->data == x ) return l;
  return NULL;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
}
```

Sau đó:

```
if ( ( p = Find( l, x ) ) != NULL )
  Solution( &l, p );
else
  printf( "Khong tim thay node!\n" );
```

1. Xóa node đầu

Cần nắm vững các bước xóa node đầu, vì nó là cơ sở cho các trường hợp còn lại.
- Phải khai báo một con trỏ `d` (kiểu `NODEPTR`) giữ lại node cần xóa.
- Con trỏ `l` "tiến lên" để lại node đầu cho con trỏ `d` giữ.
- Bây giờ có thể xóa `d`.
- Phải kiểm tra xem `l` có khác `NULL` không trước khi xóa.

```
void Solution( NODEPTR* l ) {
  if ( *l ) {
    NODEPTR d = *l;
    *l = ( *l )->next;
    free( d );
  }
}
```

Xóa node đầu nên `l` thay đổi khi gọi hàm, do đó phải truyền `l` bằng con trỏ.

2. Xóa node `p`

Xóa node không quan tâm đến thứ tự trong danh sách:
- Sao chép dữ liệu từ node đầu vào node `p`, như vậy node `p` đã bị xóa, nhưng lại có 2 node chứa nội dung giống node đầu.
- Xóa node đầu như trên.
- Phải kiểm tra xem `l` có khác `NULL` không trước khi xóa.

```
void Solution( NODEPTR* l, NODEPTR p ) {
  if ( *l ) {
    p->data = ( *l )->data;
    p = l;              /* dùng lại con trỏ p */
    *l = ( *l )->next;
    free( p );
  }
}
```

Xóa node có quan tâm đến thứ tự trong danh sách:
Theo quan điểm node bất kỳ của danh sách cũng là node đầu (node đầu của *danh sách còn lại*), ta thấy thực chất việc xóa node `p` là xóa node đầu (xem các bước xóa node đầu):
- Con trỏ giữ node để xóa là `p`.
- Con trỏ chỉ vào node cần xóa (sẽ "tiến lên"):
Trường hợp `p` là node đầu danh sách, con trỏ chỉ vào node là `l`, sẽ "tiến lên": `l = l->next`
Trường hợp `p` là node nằm giữa danh sách, phải xác định node ngay trước `p` (gọi là node `q`) bằng phương pháp "dò phía trước" (lặp cho đến khi `q->next` chỉ vào `p`). Nếu

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

có, con trỏ chỉ vào node là `q->next`, sẽ "tiến lên":
`(q->next) = (q->next)->next`.
Để đơn giản, ta dùng con trỏ tương đương: `q->next = p->next`
- Xóa node, tức giải phóng `p`.

```
void Solution( NODEPTR* l, NODEPTR p ) {
  if ( *l ) {
    if ( p == *l ) *l = ( *l )->next;
    else {
      NODEPTR q;
      for ( q = *l; q->next && q->next != p; q = q->next ) { }
      if ( q->next ) q->next = p->next;
    }
    free( p );
  }
}
```

3. Xóa đệ quy

```
void Solution( NODEPTR* l, NODEPTR p ) {
  if ( !*l ) return;
  if ( p == *l ) {
    *l = ( *l )->next;
    free( p );
    return;
  }
  Solution( &( *l )->next, p );
}
```

Khi đệ quy, lúc nào ta cũng đang xét *node đầu của danh sách* (danh sách ban đầu cũng như các *danh sách còn lại* khi đệ quy). Vì vậy, ta đưa về trường hợp xóa đầu. Có hai trường hợp dừng đệ quy: đã đến cuối danh sách; đã tìm và xóa được node.

Bài 205: (trang 60)

```
void Solution( NODEPTR* l ) {
  if ( !*l ) return;
  Solution( &( *l )->next );
  if ( *l ) {
    NODEPTR p = *l;
    *l = ( *l )->next;
    printf( "Xoa node [%d]\n", p->data );
    free( p );
    return;
  }
}
```

Ta thực hiện xóa đệ quy từng node của danh sách liên kết kể từ đầu đến cuối giống như bài trên. Sau đó chuyển giải pháp trở thành "đệ quy đầu". Khi đó đệ quy sẽ được thực hiện đến cuối danh sách liên kết, lưu các thông tin vào stack hệ thống, sau đó mới tiến hành xóa từ cuối ngược lên đầu.

Bài 206: (trang 60)

```
void Solution( NODEPTR* l, int x ) {
  if ( !*l ) return;
  if ( x == ( *l )->data ) {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    NODEPTR d = *l;
    *l = ( *l )->next;
    free( d );
    Solution( l, x );
  }
  else Solution( &( ( *l )->next ), x );
}

/* gọi trong hàm main() */
Solution( &l, k );
```

Trong trường hợp phải xóa nhiều node chứa trị `x`, khi xóa một node xong ta *không dừng đệ quy*. Tuy nhiên cần chú ý khi đệ quy tiếp:
- Trong trường hợp không xóa node đang xét, ta đệ quy tiếp với danh sách còn lại, kể từ *node kế tiếp*: `l->next`
- Trong trường hợp xóa node đang xét, ta đệ quy tiếp với danh sách còn lại, kể từ *node mới dồn lên*: `l`. Điều này giúp tránh bỏ sót các node dồn lên khi xóa, xuất hiện khi có hai node cần xóa nằm kế tiếp nhau.

Bài 207: (trang 60)

```
void Solution( NODEPTR* l, NODEPTR* p ) {
  NODEPTR t, q;
  *p = q = NULL;
  while ( *l ) {
    t = *l;
    *l = ( *l )->next;
    if ( t->data % 2 ) { t->next = *p; *p = t; }
    else               { t->next =  q;  q = t; }
  }
  *l = q;           /* dùng lại con trỏ l */
}
/* gọi trong hàm main() */
Solution( &l, &p );
OutList( l, "List chan" );
OutList( p, "List le  " );
```

Duyệt danh sách liên kết đã cho, mỗi lần duyệt bỏ lại node duyệt xong cho một con trỏ `t` (kiểu `NODEPTR`) giữ. Như vậy danh sách liên kết ban đầu bị hủy dần từng node, node bị hủy chuyển dần cho con trỏ `t` giữ.
Tùy trị chứa trong `t` là chẵn hay lẻ, chèn đầu `t` (không tạo node mới) vào một trong hai danh sách mới, đại diện bởi 2 con trỏ `p`, `q` kiểu `NODEPTR` (phải khởi tạo bằng `NULL` trước).
Khi con trỏ `l` của danh sách gốc đi đến cuối danh sách, dùng lại con trỏ này để chỉ đến một trong hai danh sách mới.
Không dùng vòng lặp `for` được, vì sau khi gán `l` cho `t`, phải cho `l = l->next` ngay trước khi `l->next` bị thay đổi do chèn `t` vào `p` hoặc `q` (`l->next` lúc này là `t->next`). Đây cũng là lý do tại sao phải dùng con trỏ tạm `t`.
Nếu không phá hủy danh sách ban đầu ta thực hiện như sau:
- Duyệt danh sách đã cho, lấy lần lượt nội dung từng node của nó.
- Tùy trị chứa trong node là chẵn hay lẻ, thực hiện thao tác chèn đầu node mới chứa trị đó vào một trong hai danh sách mới, đại diện bởi 2 con trỏ `p`, `q` kiểu `NODEPTR` (phải khởi tạo bằng `NULL` trước).

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Do khi duyệt ta chỉ lấy nội dung node và chèn có tạo node mới nên danh sách cũ không bị phá hủy.

```
void Insert( NODEPTR *l, int x ) {
  NODEPTR p = ( NODEPTR )malloc( sizeof( struct NODE ) );
  p->data = x;
  p->next = *l;
  *l = p;
}

void Solution( NODEPTR l, NODEPTR* p, NODEPTR* q ) {
  *p = *q = NULL;
  for ( ; l; l = l->next )
    ( l->data % 2 ) ? Insert( p, l->data )
                    : Insert( q, l->data );
}

/* gọi trong hàm main() */
Solution( l, &p, &q );
OutList( q, "List chan" );
OutList( p, "List le  " );
```

Chú ý lời gọi hàm `Insert()` trong `Solution()`, đầy đủ phải như sau:
`Insert( &*p, l->data )`
Nhưng `&*p` tương đương với `p` nên có thể viết tắt. Cách viết này thường thấy trong các hàm đệ quy có tham số truyền như con trỏ. Tuy nhiên tránh nhầm lẫn với trường hợp sau:
`&(*t)->left` tương đương với `&((*t)->left)`, khác hẳn với `t->left`.

Bài 208: (trang 60)

```
void Solution( NODEPTR* l, int m ) {
  NODEPTR p, q;
  int c = 0;
  p = q = *l;
  for ( ; q && c < m; q = q->next, c++ ) { }
  if ( q ) {
    for ( ; q && q->next; p = p->next, q = q->next ) { }
    q->next = *l;
    *l = p->next;
    p->next = NULL;
  }
}

/* gọi trong hàm main() */
Solution( &l, m );
```

Xem hình vẽ.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Ta cần có 2 con trỏ (kiểu `NODEPTR`): một con trỏ `p` chỉ đến node cách node cuối `m` node, một con trỏ `q` chỉ đến node cuối. Dùng 2 vòng lặp như sau:
- Cho con trỏ `p` và `q` chỉ đến node đầu, vòng lặp thứ nhất đưa con trỏ `q` rời xa con trỏ `p` một khoảng `m` node. Không cần kiểm tra `m` có lớn hơn số node của danh sách không, vì trong trường hợp đó, `q` sẽ bằng `NULL` ngay sau vòng lặp đầu và không cần chuyển node nữa.
- Trong vòng lặp thứ hai, hai con trỏ cùng di chuyển song song, khi con trỏ `q` đến node cuối là đạt yêu cầu.

Như vậy các vòng lặp chỉ có nhiệm vụ di chuyển 2 con trỏ `p`, `q` đến đúng vị trí, không thực hiện thao tác nào khác.
Tiếp theo, con trỏ (`q->next`) cuối cùng chỉ đến đầu danh sách, con trỏ `l` chỉ đến danh sách bây giờ chỉ đến `p->next`, còn `p->next` trước đây trở thành con trỏ cuối danh sách (tức thành `NULL`).

Bài 209: (trang 61)

```
void OutListR( NODEPTR l ) {
  if ( !l ) return;
  OutListR( l->next );
  printf( "[%d]", l->data );
}

NODEPTR Max( NODEPTR l ) {
  if ( l->next == NULL ) return l;
  return ( l->data > Max( l->next )->data ) ? l : Max( l->next );
}

/* gọi trong hàm main() */
printf( "List dao: " );
OutListR( l );
putchar( '\n' );
if ( l ) printf( "Tri max: %d\n", Max( l )->data );
```

Ta thường duyệt danh sách liên kết đơn để in nội dung từng node theo thứ tự lưu trữ kể từ đầu danh sách theo hai cách: dùng vòng lặp hoặc dùng đệ quy.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Nếu dùng đệ quy: in nội dung chứa trong node đầu và gọi đệ quy với danh sách còn lại (gọi là đệ quy đuôi), đệ quy chấm dứt khi danh sách còn lại rỗng (`l == NULL`).

```
void OutList( NODEPTR l ) {
  if ( !l ) return;
  printf( "[%d]", l->data );
  OutList( l->next );
}
```

Nếu gọi đệ quy trước khi in nội dung chứa trong node đầu thì mãi đến khi chấm dứt đệ quy, node đầu của danh sách xét cuối cùng (tức node cuối của danh sách đã cho) mới được in ra, cứ như thế các node sẽ được in ra tiếp theo thứ tự ngược từ cuối danh sách trở lên đầu. Đây là đặc điểm của đệ quy đầu.

Hàm đệ quy `Max()` được xây dựng dựa trên phát biểu đệ quy: *phần tử lớn nhất của danh sách* được chọn bằng cách so sánh phần tử đầu danh sách với *phần tử lớn nhất của danh sách* còn lại.

Điều kiện đầu: nếu danh sách chỉ có một phần tử thì phần tử đó cũng là phần tử lớn nhất. Hàm `Max()` không dùng với danh sách rỗng.

Bài 210: (trang 61)

```
void Solution( NODEPTR* l ) {
  NODEPTR t, p = NULL;
  while ( *l ) {
    t = *l;
    *l = ( *l )->next;
    t->next = p;
    p = t;
  }
  *l = p;
}

/* gọi trong hàm main() */
Solution( &l );
OutList( l, "List dao" );
```

Ta thực hiện không đệ quy như sau:

- Duyệt danh sách liên kết đã cho, mỗi lần duyệt bỏ lại node vừa duyệt cho một con trỏ `t` (kiểu `NODEPTR`) giữ. Như vậy danh sách liên kết ban đầu dần dần bị hủy.
- Chèn đầu `t` (không tạo node mới) vào một danh sách mới, đại diện bởi con trỏ `p` kiểu `NODEPTR` (phải khởi tạo bằng `NULL` trước). Vì *chèn đầu* nên danh sách mới sẽ có *thứ tự đảo ngược* so với danh sách cũ.
- Sau khi duyệt, con trỏ `l` đã đến cuối danh sách, dùng lại con trỏ này để chỉ đến danh sách mới.

Không thay đổi liên kết của các node, ta cũng có thể đảo ngược thứ tự các phần tử trong danh sách liên kết đơn bằng cách hoán chuyển dữ liệu của các phần tử từ hai đầu danh sách:

- Vòng lặp ngoài định vị node cần hoán chuyển dữ liệu thứ nhất (nằm ở nửa đầu danh sách), từ đầu danh sách bằng con trỏ `l`.
- Vòng lặp trong định vị node cần hoán chuyển dữ liệu thứ hai (nằm ở nửa cuối danh sách), từ cuối danh sách bằng con trỏ `p`.

Vấn đề là làm sao cho con trỏ `p` di chuyển ngược lên đầu danh sách: vòng lặp trong được "chặn dưới" bởi con trỏ `q`. Sau mỗi vòng lặp trong `p` chỉ node cuối, ta gán cho

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

q trị của p nên "chặn dưới" được dịch chuyển ngược lên và vì vậy p cũng dịch chuyển ngược lên sau mỗi vòng lặp trong.

```
#define Swap( a, b ) { int t = a; a = b; b = t; }
/* ... */
void Solution( NODEPTR l ) {
  NODEPTR p, q;
  for ( q = NULL; l != q && l->next != q; l = l->next ) {
    for ( p = l; p->next != q; p = p->next ) { }
    Swap( l->data, p->data );
    q = p;
  }
}
```

Dùng giải thuật đệ quy như sau:

- Khai báo thêm một tham số p (kiểu NODEPTR), truyền bằng con trỏ và khởi tạo bằng NULL, dùng để lưu danh sách mới được tạo ra, vì ta sẽ chèn đầu nên p thay đổi liên tục và cần lưu bằng cách truyền qua các lần gọi đệ quy.
- Vẫn chèn đầu như cách không đệ quy ở trên để đảo ngược danh sách nhưng cần dùng một con trỏ q lưu l->next lại trước khi chèn vì quá trình chèn sẽ làm thay đổi l->next.
- Gọi đệ quy để xử lý danh sách còn lại (dùng q thay cho l->next làm tham số khi gọi).
- Khi chấm dứt đệ quy trả về con trỏ p, tức danh sách đã đảo ngược.

```
NODEPTR Solution( NODEPTR l, NODEPTR* p ) {
  if ( !l ) return *p;
  else {
    NODEPTR q = l->next;
    l->next = *p;
    *p = l;
    return Solution( q, p );
  }
}

/* gọi trong hàm main() */
p = NULL;
Solution( l, &p );
OutList( p, "List dao" );
```

Bài 211: (trang 61)

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

struct NODE {
  int data;
  struct NODE * next;
};
typedef struct NODE* NODEPTR;

struct RNODE {
  NODEPTR ptr;
  struct RNODE * next;
};
typedef struct RNODE* RNODEPTR;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
NODEPTR InList() {
  int x;
  NODEPTR p;
  scanf( "%d", &x );
  if ( !x ) return NULL;
  p = ( NODEPTR )malloc( sizeof( struct NODE ) );
  p->data = x;
  p->next = InList();
  return p;
}

RNODEPTR CreateRun( NODEPTR* l ) {
  NODEPTR p;
  RNODEPTR t, r;
  r = NULL;
  while ( *l ) {
    for ( p = *l; p->next && p->data < p->next->data; p = p->next ) { }
    /* tạo một node cho "run" mới trong danh sách quản lý "run" */
    t = ( RNODEPTR )malloc( sizeof( struct RNODE ) );
    /* chèn đầu "run" vừa tìm được vào danh sách quản lý "run" */
    t->ptr = *l;
    t->next = r;
    r = t;
    *l = p->next;
    p->next = NULL;
  }
  return r;
}

void ShowRun( RNODEPTR r ) {
  NODEPTR p;
  printf( "r-" );
  if ( r->next ) printf( "+-" );
  else printf( "--" );
  for ( p = r->ptr; p; p = p->next )
    printf( "[%d]", p->data );
  printf( "[n]\n" );
  if ( r->next ) {
    for ( r = r->next; r->next; r = r->next ) {
      printf( "  |-" );
      for ( p = r->ptr; p; p = p->next )
        printf( "[%d]", p->data );
      printf( "[n]\n" );
    }
    printf( "  `-" );
    for ( p = r->ptr; p; p = p->next )
      printf( "[%d]", p->data );
    printf( "[n]\n" );
  }
}

int main() {
  NODEPTR l;

  printf( "Nhap 0 de dung: " );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  l = InList();
  printf( "List 'run':\n" );
  ShowRun( CreateRun( &l ) );
  return 0;
}
```

Sơ đồ tách các "run" được trình bày trong hình dưới:

- Ta dùng một vòng lặp với con trỏ `p` để xác định node cuối của một "run" (hoặc báo đã đến node cuối của danh sách liên kết). Khi chấm dứt vòng lặp, node đầu của "run" do `l` chỉ đến và node cuối "run" do `p` chỉ đến. Vòng lặp này nhằm mục đích di chuyển `p` nên không có thân vòng lặp.
- Chèn đầu node mới (chứa con trỏ `l`, chỉ đến một "run") vào danh sách liên kết quản lý "run" (danh sách `r`).
- Cho `p->next = NULL` để kết thúc "run" này, nhưng trước khi `p->next` thay đổi phải cho `l = p->next` để `l` chỉ đến đầu "run" mới.
- Lần lượt xử lý từng "run" cho đến khi hết danh sách liên kết.

Hàm `ShowRun()` trong bài giải viết phức tạp một chút để trình bày kết quả trực quan, bạn có thể viết đơn giản như sau:

```
void ShowRun( RNODEPTR r ) {
  NODEPTR p;
  for ( ; r; r = r->next ) {
    for ( p = r->ptr; p; p = p->next )
      printf( "[%d]", p->data );
    printf( "[n]\n" );
  }
}
```

Danh sách các "run" `r` được duyệt, với mỗi node của danh sách `r`, in danh sách "run" do node đó quản lý.

Bài 212: (trang 61)

```
NODEPTR MergeTwoRun( NODEPTR* r1, NODEPTR r2 ) {
  NODEPTR p, t1, t2;
  while ( *r1 ) {
    p = *r1;
    *r1 = ( *r1 )->next;
    p->next = NULL;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    /* chèn p vào r2 theo đúng thứ tự */
    t1 = NULL;
    t2 = r2;
    while ( t2 && t2->data < p->data ) {
      t1 = t2;
      t2 = t2->next;
    }
    if ( !t1 ) { p->next = r2; r2 = p; }
    else       { p->next = t1->next; t1->next = p; }
  }
  return r2;
}

NODEPTR MergeRuns( RNODEPTR run ) {
  RNODEPTR p;
  for ( p = run; p->next; p = p->next )
    p->next->ptr = MergeTwoRun( &p->ptr, p->next->ptr );
  return p->ptr;
}
```

Hàm `MergeTwoRun()`: dùng trộn hai "run" tăng `r1` và `r2` thành một "run" tăng (`r2`) được thực hiện theo ý tưởng "chèn từng node tách ra từ `r1` vào `r2` theo đúng thứ tự":
- Vòng lặp ngoài (với `r1`) sẽ tách từng node của `r1`, theo giải thuật xóa node đầu, chuyển cho con trỏ `p` lưu giữ.
- Với mỗi node `p`, chèn `p` đúng thứ tự vào `r2` sao cho `r2` vẫn tăng, xem bài 203 (trang 301).

Hàm `MergeRuns()`: dùng trộn `n` "run". Trong vòng `for`, mỗi lần lặp gọi hàm `MergeTwoRun()` trộn "run" `i` và "run" `i + 1` thành "run" `i + 1`, .v.v... ; cuối cùng trộn "run" `n - 1` và "run" `n` thành một "run" `n` duy nhất,

Bài 213: (trang 61)

```
void Solution( NODEPTR l ) {
  int count, max = 0;
  NODEPTR p, pos = NULL;
  for ( ; l; l = l->next ) {
    for ( count = 1, p = l->next; p; p = p->next )
      if ( p->data == l->data ) count++;
    if ( count > max ) {
      max = count;
      pos = l;
    }
  }
  printf( "[%d](%d) ", pos->data, max );
}
```

Tham khảo bài tập 75 (trang 133), thực hiện bài tập trên với mảng. Có nhiều điểm tương đồng khi thực hiện bài tập trên với danh sách liên kết:
- Với từng trị trong node phải thực hiện một vòng lặp đếm số lần xuất hiện của nó trong danh sách liên kết. Như vậy có hai vòng lặp lồng nhau: một vòng lặp duyệt (ngoài) và một vòng lặp đếm (trong).
- Số lần đếm sẽ được so sánh với một trị `max` để tuyển ra trị có số lần xuất hiện nhiều nhất.
- Với trị đã xuất hiện thì tần số đếm lần sau sẽ nhỏ hơn khi đếm trị đó lần đầu nên không ảnh hưởng đến kết quả.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài tập: Thực tế, tần suất của các trị trong danh sách liên kết có thể bằng nhau. Hãy in tất cả các trị trong danh sách liên kết có *tần suất lớn nhất và bằng nhau*.
Tham khảo bài tập 76 (trang 134). Ta cần có một số điều chỉnh:
- Dùng con trỏ `q` thay cho con trỏ `l` vì vòng lặp này phải chạy nhiều lần.
Con trỏ `q` lúc đầu chỉ node đầu tiên của danh sách (`q = pos->next` mà `p = l` nên `q = l->next`). Những lần sau `q` chỉ node ngay sau node chứa trị có tần suất lớn nhất tìm được lần trước (`q = pos->next`).
- Một biến `oldmax` được thêm để lưu trị `max` của lần tìm trước.
- Một vòng lặp `do while` bao bên ngoài để tiến hành tìm nhiều lần.

```
void Solution( NODEPTR l ) {
  int count, max, oldmax;
  NODEPTR p, q, pos;
  oldmax = 0;
  pos = l;
  do {
    max = 0;
    for ( q = pos->next; q; q = q->next ) {
      for ( count = 1, p = q->next; p; p = p->next )
        if ( p->data == q->data ) count++;
      if ( count > max ) { max = count; pos = q; }
    }
    if ( max >= oldmax ) {
      printf( "[%d](%d) ", pos->data, max );
      oldmax = max;
    }
  } while ( max == oldmax );
}
```

Bạn có thể nhận thấy giải thuật tiến hành ở trên có nhiều điểm tương tự với giải thuật được tiến hành với mảng. Điều này được nhấn mạnh trong bài 216 (trang 315).

Bài 214: (trang 61)

```
void InsertLast( NODEPTR* l, NODEPTR p ) {
  p->next = NULL;
  if ( !*l ) *l = p;
  else {
    NODEPTR t;
    for ( t = *l; t->next; t = t->next ) { }
    t->next = p;
  }
}

void Solution( NODEPTR* l, NODEPTR* p ) {
  if ( !*l ) {
    *l = *p;
    return;
  }
  if ( ( *l )->data % 2 ) {
    NODEPTR t = *l;
    *l = ( *l )->next;
    InsertLast( p, t );
    Solution( l, p );
  }
  else Solution( &( *l )->next, p );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
}

/* gọi trong hàm main() */
p = NULL;
Solution( &l, &p );
```

Bài tập được thực hiện giống như xóa đệ quy các node (xem bài 204, trang 302) chứa trị lẻ trong danh sách liên kết, để còn lại danh sách liên kết với các node chứa các trị chẵn theo đúng thứ tự. Nhưng các node chứa trị lẻ sau khi tách ra khỏi danh sách liên kết không xóa luôn, mà được *chèn cuối* vào một con trỏ `p` (kiểu `NODEPTR`, phải khởi tạo bằng `NULL`).

Thao tác chèn cuối nhằm mục đích *không thay đổi thứ tự lưu trữ* các node so với danh sách nhập. Trong lần chèn cuối đầu tiên, con trỏ `p` thay đổi; ta cần lưu thay đổi này đến khi kết thúc đệ quy bằng cách truyền con trỏ `p` như tham số qua các lần gọi đệ quy, cho dù các lần chèn cuối tiếp theo con trỏ `p` không thay đổi.

Khi kết thúc gọi đệ quy, con trỏ `l` chỉ đến cuối danh sách các trị chẵn, cho con trỏ `l` chỉ đến `p` (bằng `l = p`) để nối hai danh sách lại tạo thành danh sách kết quả.

Bài 215: (trang 62)

```
void Solution( NODEPTR l ) {
  NODEPTR p, q, t;
  for ( p = l, t = q = l->next;
        p->next && q->next;
        p = p->next, q = q->next ) {
    p->next = p->next->next;
    q->next = q->next->next;
  }
  p->next = t;
}
```

Ta thực hiện như hình sau:

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

- Khai báo hai con trỏ `p`, `q` kiểu `NODEPTR`, một dùng kết nối các node có thứ tự lẻ và một dùng để kết nối các node có thứ tự chẵn. Như vậy phải khởi tạo: `p = l` và `q = l->next`. Việc kết nối này phải *tiến hành song song*, nếu không các liên kết sẽ bị rời ra.
- Khi `p` di chuyển thì danh sách thứ tự lẻ đã có con trỏ `l` quản lý. Ta cũng dùng một con trỏ `t` kiểu `NODEPTR` để quản lý danh sách thứ tự chẵn khi `q` di chuyển. Khi duyệt hết thứ tự lẻ, cho `p->next` chỉ vào node `t` để nối danh sách thứ tự lẻ với danh sách thứ tự chẵn, hoàn thành công việc.

Nếu số node danh sách là số lẻ, do phép gán: `q->next = q->next->next;`
nên danh sách thứ tự chẵn sẽ luôn kết thúc bởi con trỏ `NULL`.

Bài 216: (trang 62)

```
#define Swap( a, b ) { int t = a; a = b; b = t; }
/* ... */
void Solution( NODEPTR l ) {
  NODEPTR p, q;
  for ( p = l; p->next; p = p->next )
    for ( q = p->next; q; q = q->next )
      if ( p->data > q->data )
        Swap( p->data, q->data );
}
```

Xem lại giải thuật sắp xếp Selection Sort trên mảng:

```
#define Swap( a, b ) { int t = a; a = b; b = t; }
/* ... */
void selectsort( int a[], int n ) {
  int i, j;
  for ( i = 0; i < n - 1; ++i )
    for ( j = i + 1; j < n; ++j )
      if ( a[i] > a[j] ) Swap( a[i], a[j] );
}
```

Thực hiện `n - 1` đợt sắp xếp (vòng lặp ngoài, chỉ số `i`), mỗi đợt sắp xếp cho vị trí `a[i]`. Mảng được sắp xếp từ đầu đến cuối. Để sắp xếp đúng vị trí `a[i]`, ta so sánh `a[i]` với các phần tử sau nó (từ `j = i + 1` trở đi), chọn ra phần tử *nhỏ nhất* đặt vào vị trí `a[i]`, bằng cách duyệt các phần tử sau `a[i]` (vòng lặp trong, chỉ số `j`), phần tử đang duyệt `a[j]` nào nhỏ hơn phần tử `a[i]` đang xét thì hoán đổi với `a[i]`.

Danh sách liên kết có những thao tác tương tự như trên mảng:

Mảng a	Danh sách liên kết l	Ý nghĩa với danh sách liên kết
`i = 0`	`p = l`	Chỉ đến đầu danh sách
`i < n - 1`	`p->next != NULL`	Chưa đến cuối danh sách
`i++`	`p = p->next`	Đến phần tử kế tiếp
`j = i + 1`	`q = p->next`	Phần tử này ngay sau phần tử kia
`a[i] > a[j]`	`p->data > q->data`	So sánh hai trị

Từ những nhận xét trên ta dễ dàng viết được giải thuật Selection Sort trên danh sách liên kết đơn chứa các trị nguyên.

Bài 217: (trang 62)

```
NODEPTR SwapTwoNode( NODEPTR p ) {
  NODEPTR t = p->next->next->next;
  p->next->next->next = p->next;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  p->next = p->next->next;
  p->next->next->next = t;
  return p->next->next;
}

void Solution( NODEPTR* l, int x ) {
  NODEPTR p, g;
  /* chèn đầu ghost node, không cần lưu trị trong ghost node */
  g = ( NODEPTR )malloc( sizeof( struct NODE ) );
  g->next = *l;
  *l = g;
  for ( p = g; p->next && p->next->next; )
    p = ( p->next->data == x ) ? SwapTwoNode( p ) : p->next;
  /* xóa đầu ghost node */
  *l = ( *l )->next;
  free( g );
}

/* gọi trong hàm main() */
printf( "Nhap k: " );
scanf( "%d", &k );
Solution( &l, k );
```

Đảo hai node kế tiếp nhau trong một danh sách liên kết được thực hiện như sau:

Cần chú ý nguyên tắc:

- Khi thay đổi một *thông tin liên kết* phải bảo đảm thông tin đó được lưu lại.

Ví dụ: trước khi thay đổi `p->next->next->next` ta lưu thông tin này vào `t`.

- Nếu không lưu *thông tin liên kết* lại trước khi thay đổi chúng thì phải bảo đảm là còn cách khác để nhận được thông tin này.

Ví dụ: node `a` do `p->next` chỉ đến, ta thay `p->next` bằng `p->next->next` được vì sau đó vẫn xác định được node `a` do con trỏ `p->next->next` chỉ đến.

- Một *thông tin liên kết* không còn ý nghĩa nữa thì có thể thay đổi mà không cần lưu lại.

Ví dụ: ở bước cuối, thông tin liên kết đến node `b` đã được `p->next` lưu nên có thể thay đổi thông tin liên kết từ node `a` trỏ đi.

Khi đảo hai node trong danh sách liên kết có các ngoại lệ sau:

- Node `a` là node cuối, khi đó `p->next->next = NULL`, ta kiểm tra và không thực hiện

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

hoán chuyển nữa.
- Node a là node đầu, ta chèn thêm node giả (ghost node) vào để đưa về trường hợp chung. Có thể nhận thấy quá trình tìm node để đảo không xét node giả này.
- Danh sách rỗng (không kể node giả), khi đó `p->next = NULL`, ta kiểm tra và không thực hiện hoán chuyển nữa.

Để đảo nhiều node chứa trị `k` yêu cầu ta cần chú ý khi đảo hai node, node chứa trị `k` chuyển về phía sau; nếu vòng lặp tiến đến node kế tiếp bằng `p = p->next` ta sẽ gặp lại node vừa đảo. Vì thế hàm `SwapTwoNode()` dùng đảo hai node được thiết kế trả về `p->next->next` để vượt qua node vừa đảo.

Bài 218: (trang 62)

```
void Swap( NODEPTR* a, NODEPTR* b ) {
  NODEPTR t = *a; *a = *b; *b = t;
}

void Solution( NODEPTR* l ) {
  NODEPTR p, p1, q, q1;
  for ( p1 = p = *l; p->next; p1 = p, p = p->next )
    for ( q1 = q = p->next; q; q1 = q, q = q->next )
      if ( p->data > q->data ) {
        Swap( &p->next, &q->next );
        ( p == *l ) ? Swap( l, &q1->next )
                    : Swap( &p1->next, &q1->next );
        Swap( &p, &q );
      }
}
```

Vấn đề được giải quyết giống bài 216 (trang 315). Tuy nhiên, ta cần hoán chuyển hai node, không phải hoán chuyển trị chứa trong chúng.

Trong danh sách liên kết, thông tin vị trí một node còn lưu trữ trong node trước nó. Vì vậy một số thao tác trên một node cần đến con trỏ quản lý node trước node đó. Để có được con trỏ này người ta thường dùng kỹ thuật "hai con trỏ nối đuôi nhau".

```
for ( p1 = p = l; p->next; p1 = p, p = p->next ) { }
```

`p` chạy từ đầu danh sách `l` đến cuối danh sách. Trước khi `p` di chuyển, dùng `p1` để lưu lại vị trí cũ của `p`, vậy `p1` luôn là con trỏ quản lý node đứng trước `p`. Chú ý nếu `p` chưa di chuyển, `p1` và `p` chỉ chung một node.

Khi hoán chuyển hai node, trước tiên hoán chuyển *thông tin liên kết đến node sau trong nó*, sau đó hoán chuyển *thông tin liên kết đến nó chứa trong node ngay trước*. Các bước cần thực hiện khi hoán chuyển hai node `p` (có node trước là `p1`) và `q` (có node trước là `q1`) trình bày trong hình trang sau:

- Hoán chuyển *thông tin liên kết chỉ đến node sau*, chứa trong node đang xét.

```
Swap( p->next, q->next );
```

- Hoán chuyển *thông tin liên kết chỉ đến node đang xét*, chứa trong node ngay trước node đang xét. Chú ý trường hợp node đang xét là node đầu.

```
if ( p == l ) Swap( l, q1->next );
else          Swap( p1->next, q1->next );
```

- Sau khi hoán chuyển *vai trò* của `p` và `q` đổi chỗ nhau nên cần hoán chuyển chúng trở lại:

```
Swap( p, q );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Hình dưới mô tả một vài trường hợp đặc biệt: hai node kế tiếp nhau (trái), một trong hai node là node đầu (phải). Bạn có thể kiểm chứng tính chính xác của cách hoán chuyển node trên.

Bài 219: (trang 62)

```
#define Swap( a, b ) { int t = a; a = b; b = t; }
/* ... */
void Solution( NODEPTR l ) {
  NODEPTR p, q;
  for ( p = NULL; p != l->next; p = q )
    for ( q = l; q->next != p; q = q->next )
      if ( q->data > q->next->data )
        Swap( q->data, q->next->data );
}
```

Xem lại giải thuật sắp xếp Bubble Sort trên mảng:

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
#define Swap( a, b ) { int t = a; a = b; b = t; }
/* ... */
void bubblesort( int a[], int n ) {
  int i, j;
  for ( i = n - 1; i > 0; --i )
    for ( j = 0; j < i; ++j )
      if ( a[j] > a[j + 1] ) Swap( a[j], a[j+1] );
}
```

Thực hiện `n - 1` đợt sắp xếp (vòng lặp ngoài, chỉ số `i`), mỗi đợt sắp xếp cho vị trí `a[i]`. Mảng được sắp xếp từ cuối đến đầu. Trong mỗi đợt sắp xếp, ta so sánh các phần tử gần nhau, hoán đổi chúng cho đúng chiều sắp xếp. Chuỗi "so sánh - hoán đổi" liên tục này sẽ dần dần đẩy phần tử `a[i]` "nổi" dần lên đúng vị trí sắp xếp. Mảng được sắp xếp từ đầu đến cuối.

Ta nhận thấy vòng lặp bên ngoài (`i`) đi ngược từ cuối mảng lên đầu. Thao tác này khó thực hiện được khi dùng danh sách liên kết đơn, chỉ có thể thực hiện trên danh sách liên kết đôi.

Tuy nhiên, do một số đặc điểm của hai vòng lặp trên, ta vẫn có thể điều khiển được một con trỏ đi ngược từ cuối danh sách liên kết đơn lên đầu danh sách bằng một thủ thuật nhỏ sau:

```
for ( p = NULL; p != l->next; p = q )
  for ( q = l; q->next != p; q = q->next )
    if ( q->data > q->next->data )
      Swap( q->data, q->next->data );
```

Vòng lặp trong (`j`) dừng khi `q->next` chỉ đến con trỏ "chặn dưới" `p` (lần đầu gán `p` bằng `NULL`). Như vậy `q` chỉ đến node trước `p`; sau mỗi vòng lặp ngoài (`i`), ta đơn giản gán cho `p` trị chứa trong `q`, kết quả là con trỏ "chặn dưới" `p` di chuyển ngược lên đầu một node.

Bài 220: (trang 62)

```
void Solution( NODEPTR* l ) {
  NODEPTR t, t1, t2;
  NODEPTR p;
  if ( *l ) {
    /*  list p có 1 phần tử  "đã sắp xếp", là phần tử đầu list l */
    p = *l;
    *l = ( *l )->next;
    p->next = NULL;
    while ( *l ) {
      /* xóa đầu l từng node, lưu node xóa vào t */
      t = *l;
      *l = ( *l )->next;
      t->next = NULL;
      /* chèn t vào danh sách "đã sắp xếp" p */
      t1 = NULL;
      t2 = p;
      while ( t2 && t2->data < t->data  ) {
         t1 = t2,
        t2 = t2->next;
      }
      if ( !t1 ) { t->next = p; p = t; }
      else       { t->next = t1->next; t1->next = t; }
    }
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    /* cập nhật lại l với p "đã sắp xếp" */
    *l = p;
  }
}
```

Giải pháp dựa trên giải thuật sắp xếp Insertion Sort trên mảng:

```
void insertsort( int a[], int n ) {
  int v, i, j;
  for ( i = 1; i < n; ++i ) {
    v = a[i];           /* lưu a[i] vào v để dồn mảng */
    j = i;
    /* dồn mảng từ a[i] ngược lên đến vị trí chèn */
    while ( a[j - 1] > v ) {
      a[j] = a[j - 1];
      j--;
    }
    a[j] = v;           /* j là vị trí chèn, đưa v vào */
  }
}
```

Thuật toán chia mảng thành hai phần;
- Phần "đã sắp xếp", nằm đầu mảng, được khởi tạo có một phần tử `a[0]`.
- Phần còn lại là phần chưa sắp xếp. Từng phần tử trong phần này sẽ được chèn đúng vị trí vào phần "đã sắp xếp".

Ta thực hiện tương tự trên danh sách liên kết:
- Tạo một danh sách "đã sắp xếp" do `p` quản lý, được khởi tạo có một phần tử là phần tử đầu danh sách `l` cần sắp xếp.
- Xóa đầu dần danh sách `l`, từng node xóa chuyển cho `t` quản lý. Chèn `t` vào danh sách *có thứ tự* `p` sao cho vẫn bảo đảm thứ tự đó, xem bài 203 (trang 301).

Bài 221: (trang 63)
Gọi `t` là con trỏ chỉ đến node cần xóa. Ta sẽ xóa node này không phải thông qua `t` mà thông qua một con trỏ khác cũng chỉ đến node cần xóa, con trỏ `d` với `d = t`.
Có ba trường hợp:
- Node cần xóa không có cây con bên phải (`t->right = NULL`)
- Node cần xóa không có cây con bên trái (`t->left = NULL`)
- Node cần xóa có đủ các cây con bên trái và bên phải, tức node cần xóa là *node trong*.

a) Node cần xóa không có cây con bên phải
Vì node cần xóa đã được giữ bởi con trỏ `d`, ta đơn giản chỉ "vượt" qua node cần xóa xuống cây con bên trái:

```
if ( t->right == NULL ) t = t->left;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Node cần xóa bây giờ đã tách khỏi cây và được quản lý bởi con trỏ `d`, có thể xóa dễ dàng:

```
free( d );
```

b) Node cần xóa không có cây con bên trái

Tương tự trường hợp trên, chỉ "vượt" qua node cần xóa xuống cây con bên phải:

```
if ( t->left == NULL ) t = t->right;
free( d );
```

c) Node cần xóa là node trong

Khi xóa node `t`, node thay thế node `t` phải bảo đảm tính chất của cây BST, nghĩa là: phải có trị lớn hơn tất cả các trị trong các node của cây con bên trái node `t`, đồng thời phải có trị nhỏ hơn tất cả các trị trong các node của cây con bên phải node `t`.

Vậy node thay thế node `t` phải là node cực phải của cây con bên trái node `t` (tức node lớn nhất trong cây con bên trái, gọi là node "thay thế trước") hoặc node cực trái của cây con bên phải node `t` (tức node nhỏ nhất của cây con bên phải, gọi là node "thay thế sau"). Ta quy ước chọn node thay thế là node cực phải của cây con bên trái node `t` (node "thay thế trước").

Có hai cách xóa một node trong:

- Cách thứ nhất, xóa bằng cách sao chép node.

Tìm node "thay thế trước" là node phải nhất của cây con bên trái node `t`. Ta bắt đầu từ node gốc của cây con bên trái (node `t->left`, nghĩa là `d = t->left`), sau đó luôn đi về bên phải (`d = d->right`) cho đến khi không còn đi tiếp được, nghĩa là khi con trỏ `d->right == NULL`.

Trong khi con trỏ `d` di chuyển, ta dùng con trỏ `p` nối đuôi con trỏ `d`, tức `p` luôn chỉ node cha của node đang xét (node đang xét do `d` chỉ).

```
for ( p = t, d = t->left; d->right; p = d, d = d->right )
  { }
```

Kết thúc vòng lặp trên `d` sẽ chỉ node "thay thế trước" và `p` là cha của `d`.

Sao chép chồng dữ liệu chứa trong node "thay thế trước" vào node cần xóa:

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
t->data = d->data;
```

Bây giờ có cây 2 node chứa dữ liệu giống nhau, ta sẽ xóa node "thay thế trước" vì node này hoặc là node lá, hoặc chỉ có một cây con bên trái nên dễ dàng xóa hơn. Node cha của node "thay thế trước" là `p`, lưu được do dùng kỹ thuật hai con trỏ nối đuôi ở bước trước. Cho `p` chỉ đến node gốc của cây bên trái node "thay thế trước":

```
p->right = d->left;
```

Node "thay thế trước" bây giờ đã tách khỏi cây và quản lý bởi con trỏ `d`, ta xóa `d`:

```
free( d );
```

Khi tìm node "thay thế trước", có một trường hợp đặc biệt là node "thay thế trước" nằm ngay sau node cần xóa. Kết quả vòng lặp tìm node "thay thế trước" như sau:

Trong trường hợp này `p = t` và `d` chính là `p->left`. Sau khi sao chép chồng dữ liệu chứa trong node "thay thế trước" vào node cần xóa:

```
t->data = d->data;
```

ta tách node "thay thế trước" ra khỏi cây bằng cách:

```
p->left = d->left;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Node "thay thế trước" bây giờ đã tách khỏi cây và quản lý bởi con trỏ `d`, ta xóa `d`:
`free( d );`
Tổng kết cách 1:

```
void RemoveNodeByCopy( NODEPTR* t ) {
  NODEPTR p, d = *t;
  if ( !( *t )->left ) *t = ( *t )->right;
  else if ( !( *t )->right ) *t = ( *t )->left;
  else {
    for ( p = *t, d = ( *t )->left; d->right;
          p = d, d = d->right ) { }
    ( *t )->data = d->data;
    if ( p == *t ) p->left = d->left;
    else           p->right = d->left;
  }
  free( d );
}
```

- Cách thứ hai, xóa node bằng ghép cây (merge).
Cách thứ hai để xóa node trong, gọn hơn cách trên, nhưng có một vài khác biệt khi tiến hành xóa node.
Tìm node "thay thế trước" là node phải nhất của cây con bên trái node `t`, không cần dùng kỹ thuật hai con trỏ nối đuôi nhau. Ta bắt đầu từ node gốc của cây con bên trái (node `t->left`, nghĩa là `p = t->left`), sau đó luôn đi về bên phải (`p = p->right`) cho đến khi không còn đi tiếp được, nghĩa là khi con trỏ `p->right == NULL`.
`for ( p = t->left; p->right; p = p->right ) { }`

Kết thúc vòng lặp trên `p` sẽ chỉ node "thay thế trước".
Cây con bên phải node cần xóa sẽ được ghép vào bên phải node "thay thế trước" thành cây con bên phải node "thay thế trước", chú ý `t->right` là con trỏ quản lý cây con bên phải của node `t`.
`p->right = t->right;`

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Như vậy, xem như node cần xóa không còn cây con bên phải (`t->right` vẫn tồn tại nhưng không còn ý nghĩa). Bạn có thể xóa dễ dàng bằng cách "vượt" qua node cần xóa xuống cây con bên trái:

```
t = t->left;
```

Node cần xóa bây giờ đã tách khỏi cây và quản lý bởi con trỏ `d`, có thể xóa dễ dàng:

```
free( d );
```

Cách này tuy đơn giản hơn nhưng dễ làm cho cây mất cân bằng, gây ảnh hưởng đến hiệu suất tìm kiếm.

Tổng kết cách 2:

```
void RemoveNodeByMerge( NODEPTR* t ) {
  NODEPTR p, d = *t;
  if ( !( *t )->left ) *t = ( *t )->right;
  else if ( !( *t )->right ) *t = ( *t )->left;
  else {
    for ( p = ( *t )->left; p->right; p = p->right ) { }
    p->right = ( *t )->right;
    *t = ( *t )->left;
  }
  free( d );
}
```

d) Xóa một node chứa trị chỉ định

Trên cơ sở thao tác xóa một node ở trên, ta tiến hành xóa một node với trị chỉ định trong cây BST bằng cách đệ quy:

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

- Nếu node đang xét chứa trị trùng với trị chỉ định thì xóa ngay node đó và dừng đệ quy.
- Nếu node đang xét chứa trị nhỏ hơn trị chỉ định thì tiến hành xóa đệ quy node chứa trị chỉ định trong cây bên trái node đang xét.
- Ngược lại, tiến hành xóa đệ quy node chứa trị chỉ định trong cây bên phải node đang xét.

```
void Remove( NODEPTR* t, int x ) {
  if ( x == ( *t )->data ) {
    RemoveNode( t );
    return;
  }
  ( x < ( *t )->data ) ? Remove( &( *t )->left, x )
                       : Remove( &( *t )->right, x );
}

/* dùng trong hàm main() */
Remove( &t, k );
```

Ta nhận thấy thao tác xóa một node (`RemoveNode()`) cách 1 tuy có ưu điểm hơn nhưng phải giải quyết trường hợp ngoại lệ và phải xác định node cha của node thay thế bằng kỹ thuật dùng hai con trỏ nối đuôi nhau. Nếu viết hàm `RemoveNode()` cách 1 ngay trong hàm đệ quy `Remove()`, cách 1 sẽ được thực hiện đơn giản như cách 2, xem chú thích trong bài:

```
void Remove( NODEPTR* t, int x ) {
  if ( x == ( *t )->data ) {
    NODEPTR p, d = *t;
    if ( ( *t )->left && ( *t )->right ) {      /* node có 2 con */
      /* tìm node thay thế như cách 2, không cần tìm node cha của node thay thế */
      for ( p = ( *t )->left; p->right; p = p->right ) { }
      /* hoán chuyển trị của node xóa và node thay thế */
      int temp = ( *t )->data;
      ( *t )->data = p->data;
      p->data = temp;
      /* trị cần xóa đã "xuống" cây con bên trái */
      /* gọi đệ quy với cây con bên trái để xóa nó */
      Remove( &( *t )->left, x );
    } else {
      *t = ( !( *t )->left ) ? ( *t )->right : ( *t )->left;
      free( d );
    }
    return;
  }
  ( x < ( *t )->data ) ? Remove( &( *t )->left, x )
                       : Remove( &( *t )->right, x );
}
```

Mục tiêu của cách xóa node này là chuyển node cần xóa có hai cây con trái phải thành node xóa chỉ có một cây con trái hoặc phải. Việc xóa node giảm thiểu khả năng mất cân bằng của cây.

Bài 222: (trang 63)

```
void LNR( NODEPTR t ) {
  if ( t ) {
    LNR( t->left );
    printf( "%d ", t->data );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    LNR( t->right );
  }
}

void NLR( NODEPTR t ) {
  if ( t ) {
    printf( "%d ", t->data );
    NLR( t->left );
    NLR( t->right );
  }
}

void LRN( NODEPTR t ) {
  if ( t ) {
    LRN( t->left );
    LRN( t->right );
    printf( "%d ", t->data );
  }
}
```

Phương pháp mô tả sau là một trong những cách duyệt cây BST một cách thủ công, giúp kiểm tra nhanh các kết quả duyệt theo chiều sâu:

- Đi theo một "đường bao" xung quanh các node cây nhị phân từ trái sang phải (nếu trong thứ tự duyệt `L` trước `R`). Khi di chuyển ta duyệt qua các node của cây 3 lần, tương ứng với ba cách duyệt cây.
- Thứ tự xuất các node được xác định như sau (chú ý các mũi tên ⇨):
  `LNR`: (inorder) xuất trị của node khi đi qua ngay "dưới" node.
  `NLR`: (preorder) xuất trị của node khi đi qua ngay "trước" (bên trái) node.
  `LRN`: (postorder) xuất trị của node khi đi qua ngay "sau" (bên phải) node.

Các cách duyệt ngược lại (backward): `RNL`, `NRL` và `RLN` cũng tiến hành tương tự.

LNR: 2 3 4 5 8 9 14 16 18

NLR: 9 3 2 5 4 8 16 14 18

LRN: 2 4 8 5 3 14 18 16 9

Ý nghĩa mỗi cách duyệt theo chiều sâu được trình bày trong bài 224 (trang 329).

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Bài 223: (trang 63)

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

struct NODE {
  int data;
  struct NODE *left, *right;
};
typedef struct NODE* NODEPTR;

/* cấu trúc dữ liệu cho queue */
struct QNODE {
  NODEPTR data;     /* dữ liệu của queue là một node của BST */
  struct QNODE *next;
};
typedef struct QNODE* QNODEPTR;

typedef struct {
  QNODEPTR qdata;       /* danh sách liên kết chứa các node của queue */
  QNODEPTR front, rear; /* con trỏ quản lý đầu và cuối queue */
} QUEUE;

/* chèn dữ liệu vào queue - chèn cuối nhờ con trỏ rear */
void InQueue( QUEUE* q, NODEPTR p ) {
  QNODEPTR h = ( QNODEPTR )malloc( sizeof( struct QNODE ) );
  h->data = p;
  h->next = NULL;
  if ( ( *q ).rear ) ( *q ).rear->next = h;
  ( *q ).rear = h;
  if ( !( *q ).front ) ( *q ).front = h;
}

/* trả về dữ liệu xuất từ queue - xóa đầu */
NODEPTR OutQueue( QUEUE* q ) {
  NODEPTR t = ( *q ).front->data;
  QNODEPTR d = ( *q ).front;
  ( *q ).front = ( *q ).front->next;
  if ( !( *q ).front ) ( *q ).rear = NULL;
  free( d );
  return t;
}

void Insert( NODEPTR* t, int x ) {
  if ( !*t ) {
    *t = ( NODEPTR )malloc( sizeof( struct NODE ) );
    ( *t )->data = x;
    ( *t )->left = ( *t )->right = NULL;
  }
  else ( x < ( *t )->data ) ? Insert( &( *t )->left, x )
                            : Insert( &( *t )->right, x );
}

void InTree( NODEPTR* t ) {
  int x;
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  printf( "Nhap 0 de dung: " );
  do {
    scanf( "%d", &x );
    if ( x ) Insert( t, x );
  } while ( x );
}

void BFS( NODEPTR t ) {
  QUEUE q;
  q.front = q.rear = NULL;
  if ( t ) InQueue( &q, t );
  while ( q.front ) {
    NODEPTR d = OutQueue( &q );
    printf( "%d ", d->data );
    if ( d->left ) InQueue( &q, d->left );
    if ( d->right ) InQueue( &q, d->right );
  }
}

int main() {
  NODEPTR t = NULL;

  InTree( &t );
  printf( "BFS: " );
  BFS( t );
  putchar( '\n' );
  return 0;
}
```

Duyệt theo chiều rộng cây BST là lần lượt duyệt từng mức (level-by-level) của cây, tại mỗi mức duyệt các node từ trái sang phải:

BFS: 9 3 16 2 5 14 18 4 8

Khi duyệt một node ta đồng thời biết được thông tin liên kết đến hai node con, thông tin này được lưu trữ vào một cấu trúc dữ liệu bổ trợ (helper data structure) nào đó để xử lý sau. Như vậy cấu trúc dữ liệu bổ trợ đó phải đáp ứng:

- Xử lý tuần tự.
- Ta đặt node cha vào cấu trúc dữ liệu đó *trước* các node con. Sau đó node cha được xử lý *trước* các node con.

Cấu trúc dữ liệu bổ trợ thích hợp là queue (hàng đợi), hoạt động theo nguyên tắc `FIFO` tuần tự, được xây dựng bằng danh sách liên kết đơn, với các thao tác:

- `InQueue()`: *chèn cuối* một node vào queue, dữ liệu trong node của queue chính là một node của cây BST.
- `OutQueue()`: *xóa đầu* một node của queue, trước khi xóa dữ liệu được lưu lại để trả về. Trong queue, thao tác nhập/xuất diễn ra ở hai đầu khác nhau của queue.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Để thực hiện nhanh thao tác chèn cuối, ta xây dựng thêm cấu trúc `QUEUE` dùng quản lý queue, trong đó đưa thêm con trỏ `rear` nhằm hỗ trợ cho thao tác chèn cuối.
Nếu dùng cấu trúc dữ liệu bổ trợ là stack, ta có kết quả như cách duyệt `NRL`.
Một cách duyệt BFS khác, không dùng cấu trúc dữ liệu bổ trợ mà dùng đệ quy:

```
#define Max( a, b ) ( ( a ) > ( b ) ? ( a ) : ( b ) )
/* ... */
void PrintLevel( NODEPTR t, int cLevel, int tLevel ) {
  if ( t )
    if ( cLevel == tLevel )
      printf( "%d ", t->data );
    else {
      PrintLevel( t->left, cLevel + 1, tLevel );
      PrintLevel( t->right, cLevel + 1, tLevel );
    }
}

int MaxLevel( NODEPTR t ) {
  if ( !t ) return 0;
  return 1 + Max( MaxLevel( t->left ), MaxLevel( t->right ) );
}

void BFS( NODEPTR t ) {
  int i;
  int maxLevel = MaxLevel( t );
  for ( i = 0; i < maxLevel; ++i )
    PrintLevel( t, 0, i );
}
```

Ta dùng hai hàm đệ quy:
- Hàm `PrintLevel( t, cLevel, tLevel )`, duyệt đệ quy cây `t` và in các node ở mức `tLevel`. `cLevel` là mức hiện hành, dùng xác định xem có đang duyệt mức `tLevel` hay không. Vì vậy khi bắt đầu duyệt, truyền tham số `cLevel` bằng `0` (mức `0`).
- Hàm `MaxLevel( t )` trả về mức sâu nhất (chiều cao) của cây `t`, cộng `1`. Bạn cũng có thể tìm chiều cao của cây trong khi chèn node để tạo cây, không nhất thiết dùng đến hàm này.

Sau đó, hàm `BFS()` chỉ đơn giản chạy vòng lặp duyệt từ mức `0` đến mức sâu nhất. Tại mỗi mức, gọi hàm `PrintLevel()` để in các phần tử thuộc mức đó.
Tuy nhiên, do cả hai hàm đệ quy trên đều hoạt động không hiệu quả nên phương pháp này không tối ưu.
Ý nghĩa cách duyệt theo chiều rộng được trình bày trong bài 225 (trang 332).

Bài 224: (trang 63)

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

struct NODE {
  int data;
  struct NODE *left, *right;
};
typedef struct NODE* NODEPTR;

void Insert( NODEPTR* t, int x ) {
  if ( !*t ) {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    *t = ( NODEPTR )malloc( sizeof( struct NODE ) );
    ( *t )->data = x;
    ( *t )->left = ( *t )->right = NULL;
  }
  else ( x < ( *t )->data ) ? Insert( &( ( *t )->left ), x )
                            : Insert( &( ( *t )->right ), x );
}

void InTree( NODEPTR* t ) {
  int x;
  printf( "Nhap 0 de dung: " );
  do {
    scanf( "%d", &x );
    if ( x ) Insert( t, x );
  } while ( x );
}

void OutTree( NODEPTR t ) {
  if ( t ) {
    printf( "%d ", t->data );
    OutTree( t->left );
    OutTree( t->right );
  }
}

void CopyTree( NODEPTR t, NODEPTR* t1 ) {
  if ( t ) {
    Insert( t1, t->data );
    CopyTree( t->left, t1 );
    CopyTree( t->right, t1 );
  }
}

void SortTree( NODEPTR t ) {
  if ( t ) {
    SortTree( t->left );
    printf( "%d ", t->data );
    SortTree( t->right );
  }
}

void RemoveTree( NODEPTR* t ) {
  if ( *t ) {
    RemoveTree( &( ( *t )->left ) );
    RemoveTree( &( ( *t )->right ) );
    printf( "%d ", ( *t )->data );
    free( *t );
  }
}

int main() {
  NODEPTR t, t1;

  t = t1 = NULL;
  InTree( &t );
  printf( "\nCay goc  : " ); OutTree( t );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  CopyTree( t, &t1 );
  printf( "\nCay copy : " ); OutTree( t1 );
  printf( "\nXuat tang: " );
  SortTree( t );
  printf( "\nXoa cay goc... " );
  RemoveTree( &t );
  if ( t ) printf( "\nCay goc rong\n" );
  return 0;
}
```

Duyệt cây BST là lần lượt đi qua tất cả các node của cây và *thực hiện một thao tác* nào đó trên mỗi node. Tùy thao tác cần thực hiện, ta chọn cách duyệt cây thích hợp cho thao tác đó:

- InOrder: vì trị các node chứa trong nhánh phải của cây BST lớn hơn trị các node chứa trong nhánh trái và node gốc chứa trị trung gian, ta dùng cách duyệt `LNR` để xuất trị chứa trong các node cây BST theo chiều *tăng dần* (`L < N < R`). Tương tự, cách duyệt `RNL` dùng xuất trị chứa trong các node cây BST theo chiều *giảm dần* (`L > N > R`). Cách duyệt này thật sự không ý nghĩa với cây tổng quát.
- PreOrder: khi muốn tạo một cây BST mới sao chép từ cây gốc, ta duyệt từng node trên cây gốc và chèn trị đọc được vào cây sao chép. Để cây sao chép có cùng cấu trúc với cây gốc, phải bảo đảm thứ tự chèn node vào cây sao chép giống như ở cây gốc, tạo node gốc trước rồi mới tạo các node con. Cách duyệt `NLR` hoặc `NRL` đáp ứng được yêu cầu này. Do đặc điểm trên, cách duyệt này còn dùng khi muốn so sánh cấu trúc của hai cây BST:

```
void CompareTree( NODEPTR t, NODEPTR t1, int* b ) {
  if ( t && t1 ) {
    if ( t->data != t1->data ) {
      *b = 0;
      return;
    }
    CompareTree( t->left, t1->left, b );
    CompareTree( t->right, t1->right, b );
  }
}

/* dùng trong hàm main() */
int b = 1;
CompareTree( t, t2, &b );
if ( b ) printf( "\nCung cau truc cay\n" );
else     printf( "\nKhac cau truc cay\n" );
```

Có thể dùng cách xuất sau để dễ dàng quan sát cấu trúc cây hơn:

```
void OutTree( NODEPTR t, char label, int d ) {
  if ( t ) {
    OutTree( t->right, 'L', d + 1 );
    printf( "%*c[%c%d]\n", 2 * d, ' ', label, t->data );
    OutTree( t->left, 'R', d + 1 );
  }
}

/* dùng trong hàm main() */
printf( "Cay goc:\n" );
OutTree( t, '*', 1 );
```

Kết quả xuất:

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
Nhap 0 de dung: 9 3 7 16 14 2 5 4 8 18 0 ↵
Cay goc:
      [L18]
    [L16]
      [R14]
  [*9]
        [L8]
      [L7]
        [R5]
          [R4]
    [R3]
      [R2]
```

Cách duyệt PreOrder còn dùng khi muốn tìm một node trong cây BST. Các phương pháp duyệt đều duyệt qua các tất cả các node trong cây, nhưng phương pháp PreOrder xử lý node ngay từ lần đầu tiên gặp node.

- PostOrder: Khi muốn xóa tất cả các node trong một nhánh cây hoặc cả cây BST, ta tiến hành xóa các node con (trái và phải) rồi mới xóa node cha của chúng. Vì vậy ta dùng cách duyệt `LRN` hoặc `RLN` cho thao tác này.

Bài 225: (trang 63)

```
void Solution( int n, NODEPTR t, int curlevel, int* sum ) {
  if ( t ) {
    if ( curlevel == n ) {
      printf( "%d ", t->data );
      *sum += t->data;
    } else if ( curlevel < n ) {
      Solution( n, t->left, curlevel + 1, sum );
      Solution( n, t->right, curlevel + 1, sum );
    }
  }
}

/* dùng trong hàm main() */
s = 0;
Solution( n, t, 0, &s );
printf( "\nTong = %d\n", s );
```

Ta duyệt đệ quy cây từ node gốc đến mức `n` yêu cầu, mức đang duyệt được xác định bởi `curlevel` được truyền như tham số. `curlevel` được khởi tạo mức `0` khi gọi hàm, khi đệ quy xuống nhánh trái hay phải `curlevel` tăng `1` đơn vị.

Nếu `curlevel` bằng mức `n` yêu cầu thì xuất nội dung node lưu trữ.

Để tính tổng các node thuộc mức `n`, ta truyền bằng con trỏ thêm một biến `sum`, những thay đổi của biến `sum` sẽ được lưu qua các lần gọi đệ quy.

Cách trên dễ hiểu nhưng `sum` được truyền bằng con trỏ nên dễ nhầm lẫn. Một phương án khác:

```
int Solution( int n, NODEPTR t, int curlevel ) {
  if ( t ) {
    if ( curlevel == n ) {
      printf( "%d ", t->data );
      return t->data;
    } else if ( curlevel < n ) {
      return Solution( n, t->left, curlevel + 1 ) +
             Solution( n, t->right, curlevel + 1 );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
    }
  }
  return 0;
}
```

Bài 226: (trang 63)

```
int Solution( int x, NODEPTR t, int level ) {
  if ( !t ) return -1;
  if ( t->data == x ) return level;
  return ( x < t->data ) ? Solution( x, t->left, level + 1 )
                         : Solution( x, t->right, level + 1 );
}

/* dùng trong hàm main() */
n = Solution( x, t, 0 );
if ( n == -1 ) printf( "Khong tim thay\n" );
else           printf( "Muc %d\n", n );
```

Ta tìm `x` trong cây `t` bằng cách duyệt cây, có hai trường hợp dừng:
- Duyệt đến node lá (`t = NULL`) mà vẫn không tìm thấy, trả về `-1` (mức không thể có trong cây). Chú ý, nếu chèn node chứa trị `x` vào, `t = NULL` chính là nơi chèn node.
- Tìm thấy node (`x = t->data`) ta trả về mức `level`. Mức `level` khởi tạo bằng `0` (mức xuất phát) khi gọi hàm và sẽ tăng `1` đơn vị khi duyệt xuống cây con trái hoặc phải.

Khi không gặp hai trường hợp trên, duyệt đệ quy xuống nhánh con trái hoặc phải tùy theo trị của `x`, chú ý tăng mức `level`.

Bài 227: (trang 64)

```
NODEPTR isMember( NODEPTR t, int x ) {
  if ( t ) {
    if ( x == t->data ) return t;
    return ( x < t->data ) ? isMember( t->left, x )
                           : isMember( t->right, x );
  }
  return NULL;
}

/* x, y phải chắc chắn thuộc cây, x < y */
NODEPTR isParent( NODEPTR t, int x, int y ) {
  if ( x == t->data || y == t->data || x < t->data && y > t->data )
    return t;
  return ( x < t->data ) ? isParent( t->left, x, y )
                         : isParent( t->right, x, y );
}

/* dùng trong hàm main() */
do {
  printf( "Nhap a, b: " );
  scanf( "%d%d", &a, &b );
  if ( a != b && isMember( t, a ) && isMember( t, b ) )
    break;
  printf( "Nhap khong hop le...\n" );
} while ( 1 );
if ( a > b ) { int temp = a; a = b; b = temp; }
printf( "Node cha: %d\n", isParent( t, a, b )->data );
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Ta *sắp xếp* trước `a < b`: giúp dễ dàng xác định được vị trí node còn lại khi xác định được một node `a` hoặc `b`.
- Nếu 2 node chứa `a` và `b` nằm trên 2 cây con trái phải của node gốc, node gốc là node cha của chúng (`TH1`).
- Nếu 2 node chứa `a` và `b` cùng nằm trong cây con bên trái hoặc bên phải của node gốc thì có hai trường hợp con:
  + Node cha là một trong hai node chứa `a` hoặc `b` (`TH2`).
  + Hai node nằm trên 2 nhánh của một *cây con* thuộc nhánh trái hoặc phải của node gốc, node cha là *node gốc của cây con* này (quy về `TH1`).

Từ node gốc `p`, ta đệ quy xuống hai nhánh trái và phải cho đến khi xảy ra một trong các điều kiện dừng đệ quy `TH1` hoặc `TH2`. Khi đó, trị trả về của hàm `isParent()` chính là node cha cần tìm.
Để bảo đảm hàm `isParent()` có điểm dừng, trị `a` và `b` phải có trong cây. Ta dùng hàm `isMember()` để xác định điều kiện này: duyệt cây theo `NLR` (tốt nhất cho tìm node). Nếu tìm được node, trả về node cần tìm; nếu không, trả về `NULL`.

Bài 228: (trang 64)

```
void PathLeft( NODEPTR t, int x ) {
  if ( t ) {
    if ( x == t->data ) return;
    ( x < t->data ) ? PathLeft( t->left, x )
                    : PathLeft( t->right, x );
    printf( "%d ", t->data );
  }
}

void PathRight( NODEPTR t, int x ) {
  if ( t ) {
    if ( x == t->data ) return;
    printf( "%d ", t->data );
    ( x < t->data ) ? PathRight( t->left, x )
                    : PathRight( t->right, x );
  }
}

/* dùng trong hàm main() */
int main() {
  NODEPTR t, p;
  int a, b;

  t = NULL;
  InTree( &t );

  do {
    printf( "Nhap a, b: " );
    scanf( "%d%d", &a, &b );
    if ( a != b && isMember( t, a ) && isMember( t, b ) )
      break;
    printf( "Nhap khong hop le...\n" );
  } while ( 1 );
  if ( a > b ) { int temp = a; a = b; b = temp; }
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  p = isParent( t, a, b );
  if ( p->data == a ) {
    PathRight( p, b );
    printf( "%d", b );
  } else if ( p->data == b ) {
    printf( "%d ", a );
    PathLeft( p , a );
  } else {
    printf( "%d ", a );
    PathLeft( p , a );
    PathRight( p->right, b );
    printf( "%d", b );
  }
  putchar( '\n' );
  return 0;
}
```

Hàm xuất đường đi từ node gốc đến một node trong cây thực chất là duyệt cây để tìm node, tốt nhất là duyệt theo `NLR`; nhưng thêm thao tác xuất từng node đã duyệt trên đường đi tới node cần tìm.

- Hàm `PathRight( t, x )`: xuất đường đi từ node gốc `p` đến node chứa trị `x` có trong cây con bên phải, node gốc chứa trị nhỏ hơn `x`.
- Hàm `PathLeft( t, x )`: xuất đường đi từ node chứa trị `x` có trong cây con bên trái đến node gốc `p` chứa trị lớn hơn `x`. Vì ta duyệt đệ quy từ node gốc xuống nên để xuất đường đi theo thứ tự ngược lại, ta dùng đệ quy đầu.

Đường đi không bao gồm node chứa `x`.

Khi xét hai node chứa `a` và `b`, `a < b` và phải có mặt trong cây BST, ta gặp các trường hợp sau đây:

- Node gốc đang xét `p` là node chứa `a`: vì `a < b` node chứa `b` nằm trong cây con bên phải node gốc `p`; gọi hàm `PathRight( p, b )` để xuất đường đi từ node gốc `p` chứa `a` đến node chứa `b`.
- Node gốc đang xét `p` là node chứa `b`: vì `a < b` node chứa `a` nằm trong cây con bên trái node gốc `p`; gọi hàm `PathLeft( p, a )` để xuất đường đi từ node chứa `a` đến node gốc `p` chứa `b`.
- Node gốc `p` không chứa `a` hoặc `b`: vì `a < b`, node `a` nằm trong cây con bên trái và node `b` nằm trong cây con bên phải cây này; gọi hàm `PathLeft( p, a )` để xuất đường đi từ node `a` đến node gốc `p`, gọi tiếp hàm `PathRight( p->right, b )` để xuất tiếp đường đi từ node phải của node gốc `p` đến node `b`. Xuất đường đi từ node phải của `p` do node gốc `p` đã xuất khi gọi hàm `PathLeft( p, a )`.

Khi giải quyết các trường hợp trên, ta cần xác định node cha gần nhất của hai node chứa trị `a` và `b`; ta dùng hàm `isParent()` cho yêu cầu này, xem bài 227 (trang 333).

Bài 229: (trang 64)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define FALSE 0
#define TRUE 1
enum eBALANCE { LEFT, EVEN, RIGHT };

struct DATA {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  int key;
  char value[10];
};

struct NODE {
  enum eBALANCE flag;
  struct DATA* pData;
  struct NODE *left, *right;
};
typedef struct NODE* NODEPTR;

void RotateRight( NODEPTR* t ) {
  NODEPTR temp = ( *t )->left;
  ( *t )->left = temp->right;
  temp->right = *t;
  *t = temp;
}

void RotateLeft( NODEPTR* t ) {
  NODEPTR temp = ( *t )->right;
  ( *t )->right = temp->left;
  temp->left = *t;
  *t = temp;
}

void Rebalance_RIGHT( NODEPTR* t ) {
  NODEPTR temp2, temp = ( *t )->right;
  switch ( temp->flag ) {
    case RIGHT: /* RR - cây con bên phải (R) có nhánh phải (R) dài */
      ( *t )->flag = EVEN;
      temp->flag = EVEN;
      RotateLeft( t );
      break;
    case LEFT:  /* RL - cây con bên phải (R) có nhánh trái (L) dài */
      temp2 = temp->left;
      switch ( temp2->flag ) {
        case EVEN:
          ( *t )->flag = EVEN;
          temp->flag = EVEN;
          break;
        case LEFT:
          ( *t )->flag = EVEN;
          temp->flag = RIGHT;
          break;
        case RIGHT:
          ( *t )->flag = LEFT;
          temp->flag = EVEN;
      }
      temp2->flag = EVEN;
      RotateRight( &temp );
      ( *t )->right = temp;
      RotateLeft( t );
  }
}

void Rebalance_LEFT( NODEPTR* t ) {
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  NODEPTR temp2, temp = ( *t )->left;
  switch ( temp->flag ) {
     /* LL - cây con bên trái (R) có nhánh trái (R) dài */
    case LEFT:
      ( *t )->flag = EVEN;
      temp->flag = EVEN;
      RotateRight( t );
      break;
    /* LR - cây con bên trái (R) có nhánh phải (R) dài */
    case RIGHT:
      temp2 = temp->right;
      switch ( temp2->flag ) {
        case EVEN:
          ( *t )->flag = EVEN;
          temp->flag = EVEN;
          break;
        case RIGHT:
          ( *t )->flag = EVEN;
          temp->flag = LEFT;
          break;
        case LEFT:
          ( *t )->flag = RIGHT;
          temp->flag = EVEN;
      }
      temp2->flag = EVEN;
      RotateLeft( &temp );
      ( *t )->left = temp;
      RotateRight( t );
  }
}
```

```
void Insert( NODEPTR* t, struct DATA* d, int* taller ) {
  /* chèn một node mới */
  if ( *t == NULL ) {
    *t = ( NODEPTR )malloc( sizeof( struct NODE ) );
    ( *t )->pData = d;
    ( *t )->flag = EVEN;
    ( *t )->left = ( *t )->right = NULL;
    *taller = TRUE;
    return;
  }
  /* trùng khóa */
  if ( d->key == ( *t )->pData->key ) {
    *taller = 0;
    return;
  }
  /* chèn trái */
  if ( d->key < ( *t )->pData->key ) {
    Insert( &( *t )->left, d, taller );
    if ( taller )        /* có khả năng làm mất cân bằng */
      switch ( ( *t )->flag ) {
        case LEFT:       /* node gốc thành //, cần cân bằng trái */
          Rebalance_LEFT( t );
          *taller = FALSE;
          break;
        case EVEN:       /* node gốc lệch trái nhưng chưa mất cân bằng */
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
          ( *t )->flag = LEFT;
          *taller = TRUE;
          break;
        case RIGHT:     /* thêm node chỉ làm cây cân bằng */
          ( *t )->flag = EVEN;
          *taller = FALSE;
        break;
      }
    return;
  }
  /* chèn phải */
  Insert( &( *t )->right, d, taller );
  if ( *taller ) {      /* có khả năng làm mất cân bằng */
    switch ( ( *t )->flag ) {
      case LEFT:        /* thêm node chỉ làm cây cân bằng */
        ( *t )->flag = EVEN;
        *taller = FALSE;
        break;
      case EVEN:        /* node gốc lệch phải nhưng chưa mất cân bằng */
        ( *t )->flag = RIGHT;
        *taller = TRUE;
        break;
      case RIGHT:       /* node gốc thành \\, cần cân bằng phải */
        Rebalance_RIGHT( t );
        *taller = FALSE;
        break;
    }
  }
  return;
}

void OutTree( NODEPTR t, int depth ) {
  int i;
  char b[3] = { '\\', '|', '/' };
  if ( t == NULL ) return;
  OutTree( t->right, depth + 1 );
  for ( i = 0; i < depth; ++i ) printf( "   " );
  printf( "(%d,%c)\n", t->pData->key, b[t->flag] );
  OutTree( t->left, depth + 1 );
}

struct DATA* Find( NODEPTR t, int key ) {
  for( ; t; ) {
    if ( t->pData->key == key ) return t->pData;
    t = ( t->pData->key > key ) ? t->left : t->right;
  }
  return NULL;
}

int main() {
  int i, taller, x;
  struct DATA* d;
  NODEPTR t = NULL;
  struct DATA data[6] = {
    { 11, "Cho" }, { 4, "Meo" }, { 12, "Heo" },
    { 3, "Cop" }, { 9, "De" }, { 6, "Ran" } };
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
  for ( i = 0; i < 6; ++i ) {
    d = ( struct DATA* )malloc( sizeof( struct DATA ) );
    d->key = data[i].key;
    strcpy( d->value, data[i].value );
    Insert( &t, d, &taller );
    OutTree( t, 0 );
    printf( "----------\n" );
  }
  printf( "Nhap khoa can tim: " );
  scanf( "%d", &x );
  if ( ( d = Find( t, x ) ) != NULL )
    printf( "[%d,%s]\n", d->key, d->value );
  else
    printf( "Khong tim thay khoa\n" );
  return 0;
}
```

Cây AVL được Adelson - Velskii và Landis giới thiệu năm 1962. Cây AVL là một cây BST cân bằng 1 cấp (1-balanced BST), điều kiện cân bằng của cây là:

$$|\ \mathtt{height}(T_L(x)) - \mathtt{height}(T_R(x))\ | \le 1, \forall x$$

Trong đó: `height()`: trả về chiều cao của cây, $T_L(x)$: cây con bên trái node `x`, $T_R(x)$: cây con bên phải node `x`

Nghĩa là chênh lệch chiều cao giữa cây con bên trái và cây con bên phải của một node `x` bất kỳ trong cây không được vượt quá `1`.

Cấu trúc dữ liệu của một node trong cây AVL cần thêm một "tác nhân cân bằng" (balancing factor) `flag` để kiểm tra cân bằng của cây:

- $\mathtt{height}(T_L(x)) > \mathtt{height}(T_R(x))$: `flag = LEFT` (minh họa bằng dấu `/`)
- $\mathtt{height}(T_L(x)) = \mathtt{height}(T_R(x))$: `flag = EVEN` (minh họa bằng dấu `-`)
- $\mathtt{height}(T_L(x)) < \mathtt{height}(T_R(x))$: `flag = RIGHT` (minh họa bằng dấu `\`)

Khi chèn một node mới vào cây AVL, ta có thể làm cây AVL mất cân bằng. Có 4 trường hợp làm cây AVL mất cân bằng:

- `RR` - cây con bên phải (`R`) có nhánh phải (`R`) dài
- `RL` - cây con bên phải (`R`) có nhánh trái (`L`) dài
- `LL` - cây con bên trái (`L`) có nhánh trái (`L`) dài
- `LR` - cây con bên trái (`L`) có nhánh phải (`R`) dài

Biến `taller` dùng trong hàm `Insert()` báo rằng cây đang "lệch" sau mỗi lần chèn node. Trong lần chèn node kế tiếp, khi nhận thấy lần chèn node trước đã làm cây "lệch", cần xác định là lần chèn node hiện tại có làm cây mất cân bằng không (hoặc làm cây cân bằng trở lại).

Sự mất cân bằng này liên quan đến 3 node: kể từ node gốc đi theo "hướng" nhánh gây mất cân bằng. Để cân bằng lại cây, người ta tái bố trí lại (trinode restructuring) sao cho 3 node này và các cây con liên quan *vẫn giữ đúng thứ tự duyệt* `LNR` (inoder, xem bài 222, trang 325).

Các hình dưới trình bày cách tái bố trí lại cây, các con trỏ trong hình giúp dễ theo dõi các hàm quay cây:

- Trường hợp `RR`:

Thao tác tái bố trí gọi là "quay trái cây" (xem hàm `RotateLeft()`) có gốc là `a`. Chú ý thứ tự duyệt `LNR (T1) a (T2) b (T3) c (T4)` không thay đổi sau khi cân bằng cây.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

- Trường hợp `RL`:

Thao tác tái bố trí thực hiện quay kép: "quay phải cây" con gốc `b` để đưa về trường hợp `RR`, sau đó "quay trái cây" gốc `a` để giải quyết trường hợp `RR`. Chú ý thứ tự duyệt `LNR (T1) a (T2) c (T3) b (T4)` không thay đổi sau khi cân bằng cây.

- Trường hợp `LL`: đối xứng với trường hợp `RR`, tái bố trí bằng cách tương tự, gọi là "quay phải cây" (xem hàm `RotateRight()`).

- Trường hợp `LR`: đối xứng với trường hợp `RL`, tái bố trí bằng cách tương tự, "quay trái cây" chuyển về `LL`, sau đó "quay phải cây" để cân bằng.

Các trường hợp `RR` và `RL` giải quyết bằng hàm `Rebalance_RIGHT()`.

Các trường hợp `LL` và `LR` giải quyết bằng hàm `Rebalance_LEFT()`.

Bài 230: (trang 65)

```
void Delete( NODEPTR* t, int d, int* shorter ) {
  if ( d == ( *t )->pData->key ) {
    if ( ( *t )->left && ( *t )->right ) {  /* có hai cây con */
      NODEPTR p;
      void* temp;
      /* tìm node thay thế trước */
      for ( p = ( *t )->left; p->right; p = p->right ) { }
      /* hoán chuyển dữ liệu node cần xóa với node thay thế trước */
      temp = ( *t )->pData;
      ( *t )->pData = p->pData;
      p->pData = temp;
      /* gọi đệ quy xóa node thay thế (hiện mang khóa d) trong cây trái */
      Delete( &( *t )->left, d, shorter );
      /* cân bằng cây sau khi xóa node */
      if ( shorter )
        switch ( ( *t )->flag ) {
          case RIGHT:   /* xóa node nhánh phải: cây mất cân bằng */
            Rebalance_RIGHT( t );
            break;
          case EVEN:    /* xóa node nhánh phải: node gốc lệch phải */
```

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí


```
            ( *t )->flag = RIGHT;
            *shorter = FALSE;
            break;
          case LEFT:    /* xóa node nhánh phải: cây cân bằng trở lại */
            ( *t )->flag = EVEN;
            *shorter = TRUE;
        }
    } else {    /* trường hợp chỉ có một cây con, xóa node tại đây */
      NODEPTR dnode = *t;
      *t = ( !( *t )->left ) ? ( *t )->right : ( *t )->left;
      free( dnode->pData );
      free( dnode );
      *shorter = TRUE;
    }
  } else if ( d < ( *t )->pData->key ) {   /* xóa bên nhánh trái */
    Delete( &( *t )->left, d, shorter );
    /* cân bằng cây sau khi xóa node */
    if ( shorter )
      switch ( ( *t )->flag ) {
        case RIGHT:
          Rebalance_RIGHT( t );
          break;
        case EVEN:
          ( *t )->flag = RIGHT;
          *shorter = FALSE;
          break;
        case LEFT:
          ( *t )->flag = EVEN;
          *shorter = TRUE;
    }
  } else {   /* xóa bên nhánh phải */
    Delete( &( *t )->right, d, shorter );
    /* cân bằng cây sau khi xóa node */
    if ( shorter )
      switch ( ( *t )->flag ) {
        case LEFT:
          Rebalance_LEFT( t );
          break;
        case EVEN:
          ( *t )->flag = LEFT;
          *shorter = FALSE;
          break;
        case RIGHT:
          ( *t )->flag = EVEN;
          *shorter = TRUE;
      }
  }
}
```

Áp dụng cách xóa node trong cây BST trình bày cuối bài tập 221 (trang 320). Kết quả node cần xóa được chuyển thành node thay thế chỉ có một cây con.

Giống như biến `taller` trong hàm `Insert()`, biến `shorter` truyền trong hàm `Delete()` báo rằng cây đang "lệch" sau mỗi lần xóa node, dùng xác định việc xóa node có làm cây mất cân bằng không để tiến hành cân bằng lại cây sau khi xóa node.

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

TÀI LIỆU THAM KHẢO

(Sắp xếp theo năm xuất bản)

[1] ***Programming Language - C. ANSI X3.159-1989 aka ISO 9899-1990*** - American National Standard for Information Systems.

[2] Kernighan, Brian W. & Ritchie, Dennis M. - ***The C Programming Language*** - Second Edition - Prentice Hall, 1988. ISBN 0-131-10370-9

[3] Kochan, Stephen G. & Wood, Patrick H. - ***Topics in C Programming*** - Third Edition - John Wiley & Sons, 1991. ISBN 0-471-53404-8

[4] Schildt, Herbert - ***A Book on C: Programming in C*** - Fourth Edition - McGraw Hill/Osborne Media, 1995. ISBN 0-078-82101-0 (tiếng Anh).

[5] Johnsonbaugh, R. & Kalin M. - ***Applications Programming in ANSI C*** - Third Edition - MacMillan, 1996. ISBN 0-023-61141-3

[6] Summit, Steve & Lafferty, Deborah - ***C Programming FAQs: Frequently Asked Questions*** - Addison Wesley, 1996. ISBN 0-201-84519-9

[7] Kelley, Al & Pohl, Ira - ***C: The Complete Reference*** - Third Edition - Addison Wesley, 1997. ISBN 0-078-82101-0

[8] Cassgne, Bernage - ***Introduction au Language C (norme ISO/ANSI)*** - Université Joseph Fourier & CNRS, 1998 (tiếng Pháp).

[9] P.S. Deshpande & O.G. Kakde - ***C & Data Structures*** - Charles River Media, 2004. ISBN 1-584-50338-6

[10] Ivor Horton - ***Beginning C*** - Fifth Edition - Apress, 2013. ISBN 978-1-4302-4881-1

[11] Jeri R. Hanly, Elliot B. Koffman - ***Problem Solving and Program Design in C*** - Seventh Edition - Pearson Education, Inc., 2013. ISBN 0-13-293649-6

[12] Deitel, H.M. & Deitel, P.J. - ***C How to Program*** - Seventh Edition - Prentice Hall, 2013. ISBN 0-13-299044-X

[13] Stephen Prata - ***C Primer Plus*** - Sixth Edition – Pearson Education, Inc., 2014. ISBN 0-321-92842-3

[14] Tony Crawford, Peter Prinz - ***C: In a Nutshell*** - Second Edition - O'Reilly, 2016. ISBN 978-1-491-90475-6

TopTaiLieu.Com | Chia Sẻ Tài Liệu Miễn Phí

Mục lục

Lời nói đầu 1
Hướng dẫn sử dụng sách 2
Phần bài tập
Khái niệm cơ bản - Toán tử - Cấu trúc lựa chọn - Cấu trúc lặp 3
Mảng 17
Mảng nhiều chiều 25
Chuỗi 35
Đệ quy 41
Structure - Union - Bit Field 46
Tập tin 49
Các vấn đề khác 56
Cấu trúc dữ liệu 58
Phần bài giải
Khái niệm cơ bản - Toán tử - Cấu trúc lựa chọn - Cấu trúc lặp 66
Mảng 113
Mảng nhiều chiều 146
Chuỗi 181
Đệ quy 217
Structure - Union - Bit Field 240
Tập tin 252
Các vấn đề khác 279
Cấu trúc dữ liệu 296
Tài liệu tham khảo 342