Bài tập về "Danh sách liên kết đơn"

03/06/2014 16:28 qn 20302 9

Các bài tập cơ bản về "danh sách liên kết đơn" (linked list) được sắp xếp, bố trí từ dễ đến khó dần nhằm giúp các bạn chưa vững (thậm chí mất căn bản) về cấu trúc dữ liệu kiểu danh sách liên kết trong ngôn ngữ C có thể tiếp cận dễ dàng.

MỤC LỤC

Một số vấn đề cơ bản về "danh sách liên kết đơn"

Để làm quen với các thao tác cơ bản trên danh sách liên kết, ta thực hiện trên một danh sách đơn giản,

Bài viết gốc

Học C

Chuẩn bị

Cài đặt Visual C++ qua web hoặc qua file ISO
Đăng ký sử dụng VS Express lâu dài
Tạo ứng dụng VC++ đơn giản nhất để học C
Gỡ rối (debug) trong Visual Studio

Bài thực hành

Kiểu char
Công thức Heron
Xoay tam giác

Bài tập

Nhập xuất dữ liệu
Thuật toán, lưu đồ
Cấu trúc rẽ nhánh
Cấu trúc lặp
Hàm & Đệ quy
Mảng một chiều

Câu hỏi thường gặp

Cài đặt, sử dụng Visual C++
Lập trình trên Visual C++
Thông báo lỗi thường gặp trong Visual C++
Hỏi bài qua Teamviewer
Đồng bộ & chia sẻ tập tin qua DropBox
Làm các bài tập C

đó là DANH SÁCH SỐ NGUYÊN (mỗi nút chỉ chứa một số nguyên). Chúng ta sẽ tìm hiểu các phần sau:

- Khai báo danh sách (cấu trúc dữ liệu)
- Cấp phát và giải phóng vùng nhớ một nút.
- Thêm một nút vào đầu danh sách.
- Thêm một nút vào cuối danh sách.
- Duyệt danh sách.
- Xóa một nút trong danh sách.

1. **Khai báo danh sách (cấu trúc dữ liệu)**

   Trước tiên là khai báo các thư viện cần thiết:

```
#include <stdio.h>    //để sử dụng hàm: printf, scanf
#include <conio.h>    //để sử dụng hàm: getch
#include <stdlib.h>   //để sử dụng hàm: malloc (cấp phát vùng nhớ cho 1 nút)
```

   Có nhiều cách để khai báo cấu trúc danh sách liên kết đơn.

Cách 1. Struct đơn thuần

```
struct Nut
{
    int GiaTri;
    struct Nut *Tiep;
};

struct Nut *dau, *cuoi;
```

Với cách khai báo trên, `Nut` chỉ là một `struct` đơn thuần (chứ không phải một kiểu), do vậy mỗi khi khai báo biến nào đó có kiểu liên quan đến cấu trúc `Nut` thì phải có từ khóa `struct` phía trước (ví dụ: `struct Nut *p`). Cách này ít khi được sử dụng.

Cách 2. Khai báo KIỂU MỚI (typedef)

```
typedef struct SoNguyen
{
    int GiaTri;
    struct SoNguyen *Tiep;
} Nut;

Nut *dau, *cuoi;
```

Với cách khai báo này, song song với việc khai báo cấu trúc `SoNguyen`, ta còn định nghĩa thêm một

"kiểu" (type) mới có tên là `Nut` (chính là kiểu có cấu trúc `SoNguyen`). Do vậy, sau này mỗi khi khai báo biến nào đó có kiểu liên quan đến cấu trúc `SoNguyen` thì thay vì khai báo `struct SoNguyen *p` ta khai báo đơn giả: `Nut *p`. Nghĩa là `Nut` được sử dụng như một kiểu, tương tự như kiểu `int` hay `float`. Một chú ý là trường `Tiep` chưa thể khai báo `Nut *Tiep` vì lúc này C chưa biết `Nut` là một kiểu, do vậy phải khai báo là `struct SoNguyen *Tiep`.

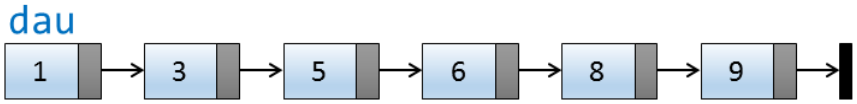
Cách 3. TroNut

```
typedef struct SoNguyen
{
    int GiaTri;
    struct SoNguyen *Tiep;
} Nut;
typedef Nut *TroNut;
TroNut dau, cuoi;
```

Với cách khai báo này, kiểu `TroNut` là một kiểu con trỏ trỏ đến một nút trong danh sách. Nghĩa là, thay vì khai báo `Nut *dau, *cuoi`, ta khai báo `TroNut dau, cuoi`.

2. Nắm vững cách biểu diễn danh sách liên kết

Ta có thể biểu diễn danh sách liên kết đơn như sau:

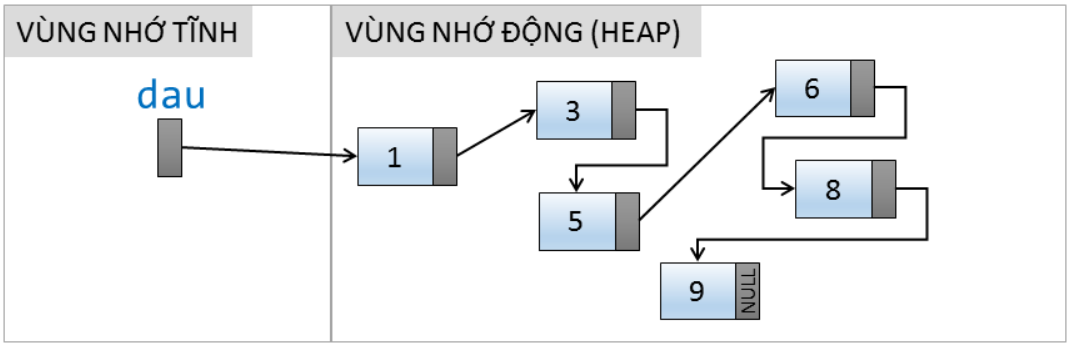


Tuy nhiên, hình ảnh trên chỉ là biểu diễn "ngắn gọn". Để nắm rõ, ta cần phân biệt hai vùng nhớ: vùng nhớ TĨNH (kích thước nhỏ) và vùng nhớ ĐỘNG (thường gọi là "heap", kích thước lớn).

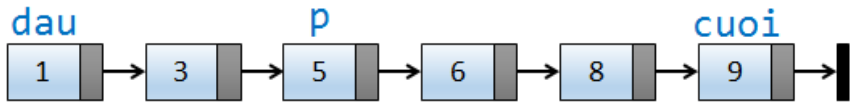
Các biến khai báo toàn cục, trong chương trình con hay trong tham số của chương trình con là những biến tĩnh, nằm ở vùng nhớ tĩnh (ví dụ: `int x; float y; char z; short *a;`). Ở đây ta tạm "lơ" qua nội tình bên trong vùng nhớ tĩnh. Các biến tĩnh khi khai báo thì đã được cấp phát vùng nhớ.

Vùng nhớ động chứa các "biến động". Các biến động không có tên mà được quản lý bởi các biến con trỏ (pointer). Chú ý, biến con trỏ nằm trong vùng nhớ tĩnh!

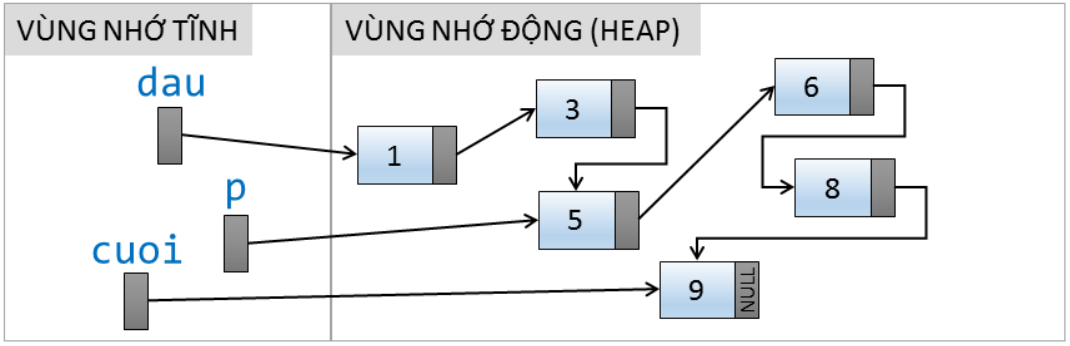
Các "nút" trong danh sách là các "biến động" nằm trong HEAP, còn những con trỏ khai báo trong chương trình chính, ví dụ `Nut *dau, *cuoi, *p` là những "biến tĩnh".



Và cũng để đơn giản, trong trường hợp có nhiều con trỏ trỏ đến các nút, ta có thể biểu diễn như sau:



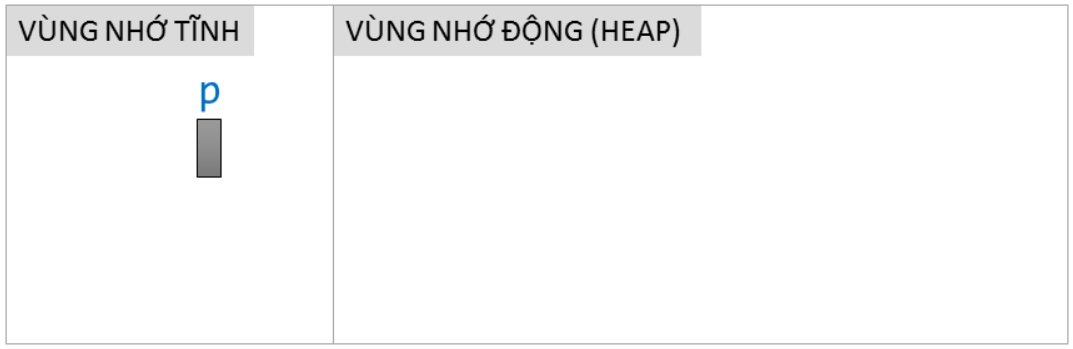
Nghĩa là con trỏ `dau` trỏ đến nút đầu tiên của danh sách (nút có giá trị là 1), con trỏ `p` trỏ đến nút có giá trị 5, và con trỏ `cuoi` trỏ đến nút cuối cùng của danh sách (nút có giá trị là 9). Tuy nhiên, chính xác hơn sẽ là thế này:



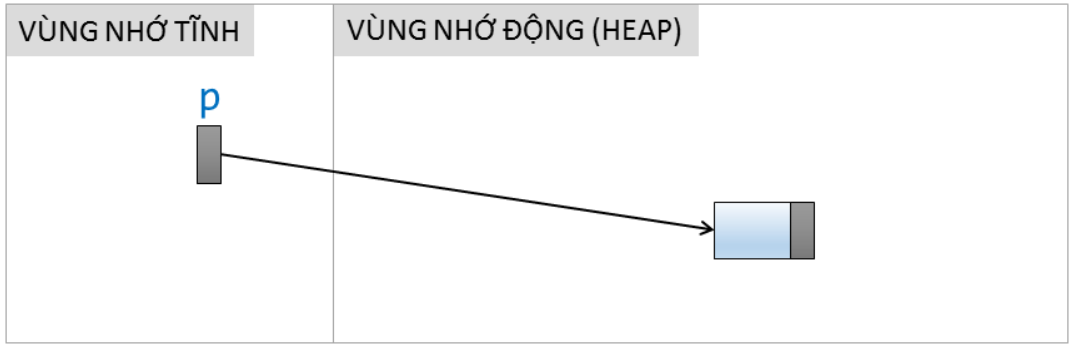
3. Cấp phát vùng nhớ cho một nút

```
Nut *p = (Nut *)malloc(sizeof(Nut));
```

Ta có thể hình dung việc "cấp phát vùng nhớ cho p" như 2 hình dưới đây. Trước khi cấp phát, con trỏ p không trỏ đến nút nào:

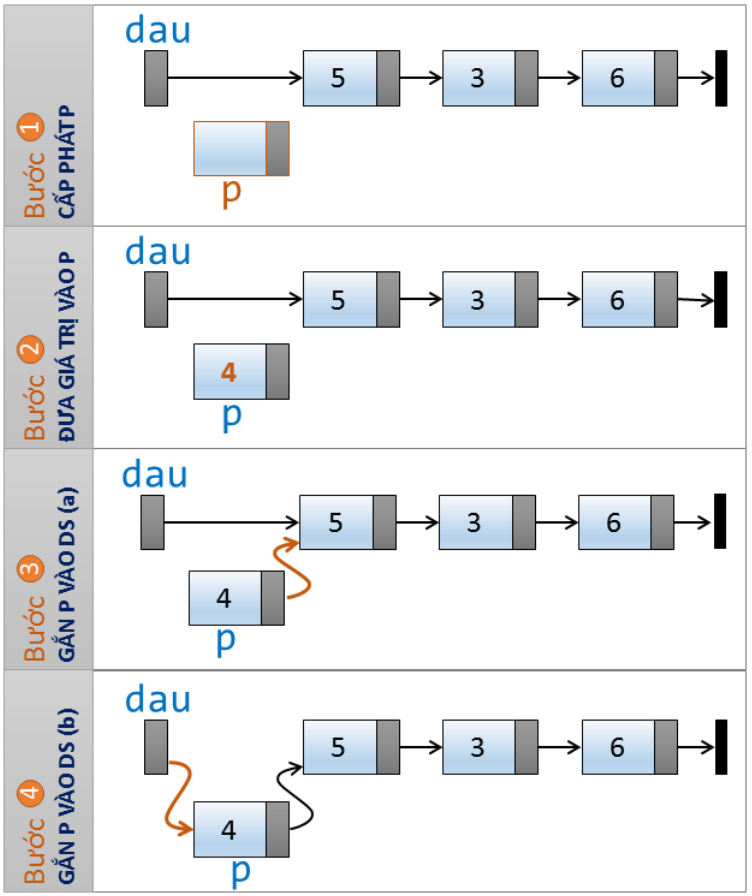


Sau khi cấp phát, nó trỏ đến một nút (biến động) ở vùng nhớ động (heap):



4. **Thêm một nút vào đầu danh sách**

Giả sử danh sách ban đầu có 3 nút có giá trị lần lượt là 5, 3 và 6. Để thêm nút có giá trị 4 vào đầu danh sách ta cần thực hiện lần lượt 4 bước sau:



Đầu tiên (bước 1) ta phải cấp phát vùng nhớ cho nút mới này và cho con trỏ `p` trỏ đến (quản lý), sau đó (bước 2) sẽ đưa giá trị vào nút. Thao tác cuối cùng là gắn nút mới này vào đầu danh sách bằng cách (bước 3) cho nút này quản lý nút đầu tiên của danh sách (`p->Tiep=dau`), tiếp đến (bước 4) cho con trỏ `dau` trỏ đến nút mới (nút mà `p` cũng đang trỏ). Chú ý là bước 3 phải thực hiện trước bước 4 vì rằng nếu bước 4 thực hiện trước (con trỏ `dau` trỏ đến nút mới ngay) thì danh sách sẽ bị "lạc lối", kiểu như "chuyển công tác" mà chưa bàn giao công việc cho người thay thế! (người thay thế ở đây là con trỏ `p->Tiep`.

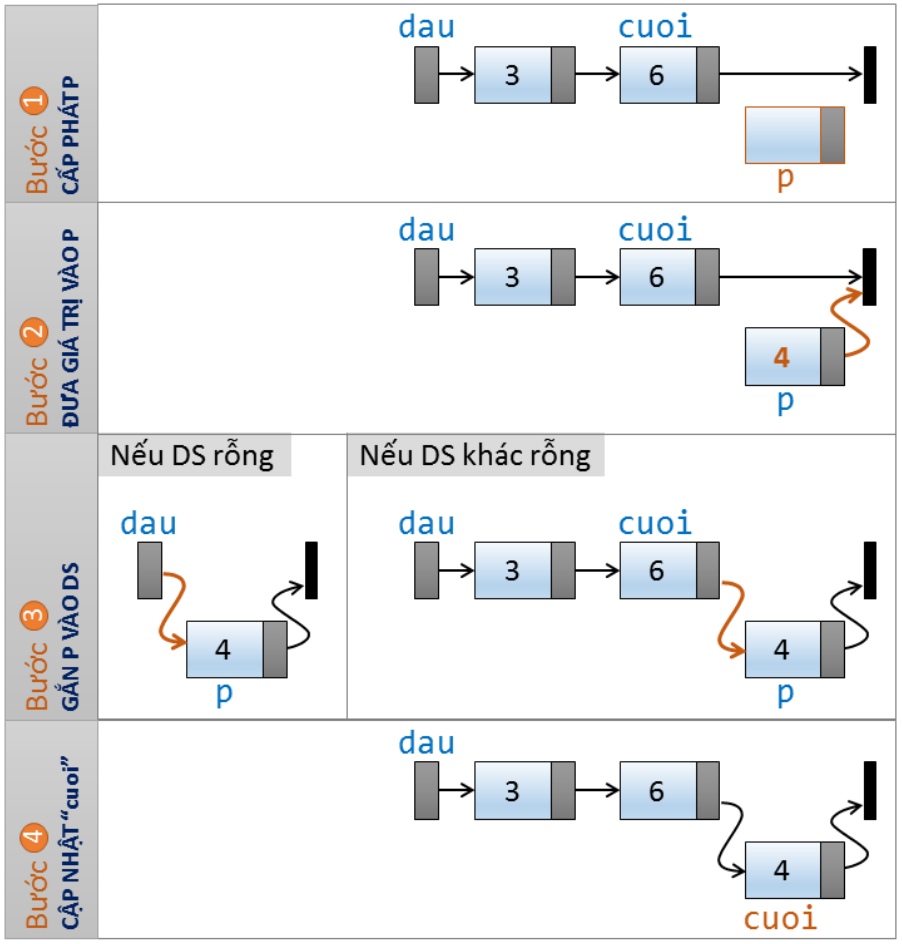
Code:

```
void themDau(int giatri)
{
    Nut *p = (Nut *)malloc(sizeof(Nut));    //1. Khai báo và cấp phát vùng nhớ cho p
    p->GiaTri = giatri;                     //2. Đưa giá trị vào p
    p->Tiep = dau;                           //3. Gắn p vào đầu danh sách
```

```
6       dau = p;                        //4. Cập nhật con trỏ dau
7   }
```

5. Thêm một nút vào cuối danh sách

Có nhiều cách để thêm một nút mới vào cuối danh sách. Dưới đây là cách có sử dụng thêm con trỏ `cuoi` (luôn trỏ đến phần tử cuối cùng của danh sách). Nếu không có con trỏ này, trước mỗi lần thêm nút mới ta phải "duyệt" toàn bộ danh sách để tìm được con trỏ trỏ đến phần tử cuối cùng (chính là con trỏ `cuoi`). Khi đã có được con trỏ `cuoi`, để gắn nút mới vào danh sách ta chỉ việc gắn nút này vào ngay sau nút được trỏ bởi `cuoi`.



Code:

```
void themCuoi(int giatri)
{
    Nut *p = (Nut *)malloc(sizeof(Nut));    //1. Khai báo và cấp phát vùng nhớ cho p
    p->GiaTri = giatri;                     //2. Đưa giá trị vào p
    p->Tiep = NULL;
    if (dau == NULL)                        //3. Gắn p vào cuối danh sách
        dau = p;
    else
        cuoi->Tiep = p;
    cuoi = p;                               //4. Cập nhật con trỏ cuoi
}
```

6. Duyệt danh sách (để hiển thị, để đếm, để tính toán,...)

Để duyệt danh sách, ta cần thêm một con trỏ p. Ban đầu p trỏ đến nút đầu tiên (p=dau). Ta sẽ "lặp" việc di chuyển p cho đến khi hết danh sách (p=NULL). Đoạn mã cho việc duyệt danh sách có thể viết như sau:

```
Nut *p = dau;
    while (p != NULL)
    {
        //DUYỆT nút p
        p = p->Tiep;
    }
```

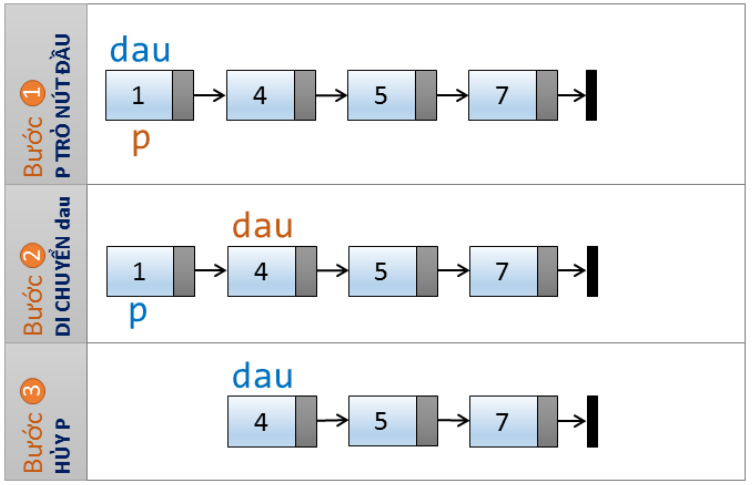
Có nhiều bài toán cần thao tác duyệt danh sách, ví dụ: hiển thị danh sách, đếm số nút trong danh sách, đếm số nút chẵn trong danh sách,...

```
void hienThiDS() //hiển thị danh sách
{
    Nut *p = dau;
    while (p != NULL)
    {
        printf("%d ", p->GiaTri);
        p = p->Tiep;
    }
}

int soNut() //đếm số nút trong danh sách
{
    Nut *p = dau; int dem = 0;
    while (p != NULL)
    {
        dem++;
        p = p->Tiep;
    }
    return dem;
}

int soNutChan() //đếm số nút chẵn trong danh sách
```

```
{
    Nut *p = dau; int dem = 0;
    while (p != NULL)
    {
        if (p->GiaTri % 2 == 0) dem++;
        p = p->Tiep;
    }
    return dem;
}
```

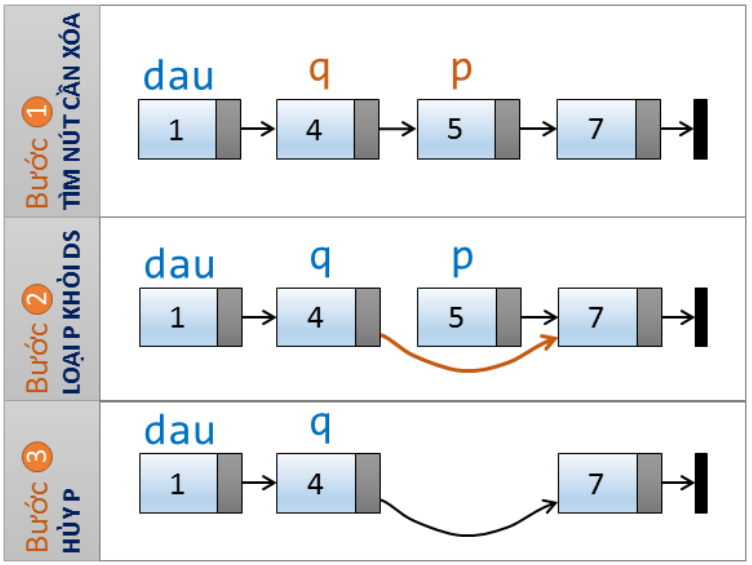
7. Xóa một nút (thỏa yêu cầu nào đó) ra khỏi danh sách

Ta cần phân biên hai trường hợp: nút cần xóa là nút đầu tiên hoặc không là nút đầu tiên trong danh sách. Nếu nút cần xóa là nút đầu tiên thì ta thực hiện như sau:




```
p = dau;
dau = dau->Tiep;
free(p);
```

Nếu nút cần xóa không là nút đầu tiên thì ta thực hiện như sau:




```
q = dau;
p = dau->Tiep;
while ([tìm chưa hết] && [tìm chưa thấy])
{
    q = p;
    p = p->Tiep;
}
if (p != NULL)
{
    q->Tiep = p->Tiep;
    free(p);
}
```

Kết hợp 2 trường hợp trên, ta có mã hoàn chỉnh của hàm xóa một nút trong danh sách:

```
void xoaNut(int giatri) //xóa MỘT nút trong danh sách có giá trị là giatri
{
    Nut *p, *q;
    if (dau == NULL) return;
    if (dau->GiaTri == giatri)
    {
        p = dau;
        dau = dau->Tiep;
        free(p);
    }
    else
    {
        q = dau;
        p = dau->Tiep;
        while (p != NULL && p->GiaTri != giatri)
        {
```

```
            q = p;
            p = p->Tiep;
        }
        if (p != NULL)
        {
            q->Tiep = p->Tiep;
            free(p);
        }
    }
}
```

8. Chương trình hoàn thiện (minh họa các thao tác vừa trình bày ở trên)

```
#include <stdio.h>    //để sử dụng hàm: printf, scanf
#include <conio.h>    //để sử dụng hàm: getch
#include <stdlib.h>   //để sử dụng hàm: malloc (cấp phát vùng nhớ cho 1 nút)

typedef struct NutSoNguyen
{
    int GiaTri;
    struct NutSoNguyen *Tiep;
} Nut;

Nut *dau, *cuoi;

void themDau(int giatri)
{
    Nut *p = (Nut *)malloc(sizeof(Nut));    //1. Khai báo và cấp phát vùng nhớ cho p
    p->GiaTri = giatri;                     //2. Đưa giá trị vào p
    p->Tiep = dau;                           //3. Gắn p vào đầu danh sách
    dau = p;                                //4. Cập nhật con trỏ dau
}

void themCuoi(int giatri)
{
    Nut *p = (Nut *)malloc(sizeof(Nut));    //1. Khai báo và cấp phát vùng nhớ cho p
    p->GiaTri = giatri;                     //2. Đưa giá trị vào p
    p->Tiep = NULL;
    if (dau == NULL)                        //3. Gắn p vào cuối danh sách
        dau = p;
    else
        cuoi->Tiep = p;
    cuoi = p;                               //4. Cập nhật con trỏ cuoi
}

void hienThiDS()
{
    Nut *p = dau;
    while (p != NULL)
    {
        printf("%d ", p->GiaTri);
        p = p->Tiep;
    }
}
```

```
42
43  int soNut()
44  {
45      Nut *p = dau; int dem = 0;
46      while (p != NULL)
47      {
48          dem++;
49          p = p->Tiep;
50      }
51      return dem;
52  }
53
54  int soNutChan()
55  {
56      Nut *p = dau; int dem = 0;
57      while (p != NULL)
58      {
59          if (p->GiaTri % 2 == 0) dem++;
60          p = p->Tiep;
61      }
62      return dem;
63  }
64
65  void xoaNut(int giatri) //xóa MỘT nút trong danh sách có giá trị là giatri
66  {
67      Nut *p, *q;
68      if (dau == NULL) return;
69      if (dau->GiaTri == giatri)
70      {
71          p = dau;
72          dau = dau->Tiep;
73          free(p);
74      }
75      else
76      {
77          q = dau;
78          p = dau->Tiep;
79          while (p != NULL && p->GiaTri != giatri)
80          {
81              q = p;
82              p = p->Tiep;
83          }
84          if (p != NULL)
85          {
86              q->Tiep = p->Tiep;
87              free(p);
88          }
89      }
90  }
91
92  int main()
93  {
94      dau = NULL; cuoi = NULL;
95      themCuoi(3);
96      themCuoi(2);
97      themCuoi(4);
```

```
    themCuoi(7);
    themCuoi(8);
    printf("\nDanh sach ban dau: "); hienThiDS();

    printf("\nDanh sach co %d nut, trong do co %d nut co gia tri chan.", soNut(), soNutChan());

    xoaNut(4);
    printf("\nDanh sach sau khi xoa nut co gia tri 4: "); hienThiDS();
    getch();
    return 0;
}
```

Bài 1. Danh sách ĐIỂM SINH VIÊN

Người ta quản lý điểm của các sinh viên trong lớp bằng cách sử dụng một danh sách liên kết đơn (mà ta gọi là **danh sách điểm**) với nút đầu được trỏ bởi con trỏ `dau`. Mỗi nút của danh sách điểm là một bản ghi gồm 3 trường: trường `HoTen` để lưu họ tên sinh viên (là một chuỗi có tối đa 30 ký tự), trường `Diem` lưu điểm của sinh viên và trường `Tiep` lưu địa chỉ của nút tiếp theo.

Xây dựng chương trình gồm các hàm sau:

- `themDau` để thêm một nút vào đầu danh sách.
- `themCuoi` để thêm một nút vào cuối danh sách.
- `taoDS` để tạo ds với thông tin được nhập từ bàn phím (dừng khi nhập họ tên là một chuỗi rỗng).
- `hienThiDS` để hiển thị danh sách (cách trình bày như ở ví dụ phía dưới).
- `soSvThiLai` để đếm xem trong danh sách có bao nhiêu sinh viên thi lại (điểm ≤ 5).
- `hienThiSvDiemMax` để hiển thị (các) sinh viên có điểm cao nhất
- `timTheoTen` để tìm kiếm và hiển thị (các) sinh viên theo tên.
- `main` để thử nghiệm các hàm vừa xây dựng ở trên.

```
Nhập họ tên sinh viên: Le Van Huan
Nhập điểm: 1

Nhap ho ten sinh vien: Phan Cong Minh
Nhập điểm: 9

Nhap ho ten sinh vien: Tran Thi Lanh
```

```
Nhập điểm: 4

Nhap ho ten sinh vien: Mai Tien Huan
Nhập điểm: 9

Nhập họ tên sinh viên:

Danh sách sinh viên vừa nhập:
- Le Van Huan (1 diem)
- Phan Cong Minh (9 diem)
- Tran Thi Lanh (4 diem)
- Mai Tien Huan (9 diem)

Có 2 sinh viên phải thi lại.

Những sinh viên có điểm tối đa (9 điểm):
- Phan Cong Minh (9 diem)
- Mai Tien Huan (9 diem)

Những sinh viên có tên 'Huan':
- Le Van Huan (1 diem)
- Mai Tien Huan (9 diem)
```

1. **Khai báo danh sách (cấu trúc dữ liệu)**

Khai báo thư viện cần thiết:

```
#include <stdio.h>    //printf, scanf
#include <conio.h>    //getch
#include <stdlib.h>   //malloc
#include <string.h>   //strcpy
```

Có nhiều cách để khai báo cấu trúc danh sách liên kết đơn. Dưới đây là 2 cách thông dụng. Vì đây là bài tập ban đầu nên để đơn giản khi xây dựng các hàm, ta đưa phần khai báo con trỏ quản lý danh sách

(dau, cuoi) ra ngoài hàm main, nghĩa là hai biến dau và cuoi là những biến toàn cục (bất cứ hàm nào trong chương trình cũng có thể truy xuất được).

Cách 1. Không khai báo kiểu mới (typedef)

```
struct SinhVien
{
    char HoTen[30];
    float Diem;
    struct SinhVien *Tiep;
};

struct SinhVien *dau, *cuoi;
```

Cách 2. Khai báo kiểu mới (typedef)

```
typedef struct SinhVien
{
    char HoTen[30];
    float Diem;
    struct SinhVien *Tiep;
} Nut;

Nut *dau, *cuoi;
```

2. Thêm một sinh viên vào đầu danh sách

Code:

```
void themDau(char hoten[], float diem)      //thêm 1 nút vào đầu danh sách
{
    Nut *p = (Nut *)malloc(sizeof(Nut));    //khai báo và cấp phát vùng nhớ cho p
    strcpy(p->HoTen, hoten);                //đưa dữ liệu (hoten, diem) vào p
    p->Diem = diem;
    p->Tiep = dau;                          //gắn p vào đầu danh sách
    dau = p;
}
```

3. Thêm một sinh viên vào cuối danh sách

Code:

```
void themCuoi(char hoten[], float diem)     //thêm 1 nút vào cuối danh sách
{
    Nut *p = (Nut *)malloc(sizeof(Nut));    //khai báo và cấp phát vùng nhớ cho p
    strcpy(p->HoTen, hoten);                //đưa dữ liệu (hoten, diem) vào p
    p->Diem = diem;
    p->Tiep = NULL;                         //gắn p vào cuối danh sách
    if (dau == NULL)
        dau = p;
```

```
    else
        cuoi->Tiep = p;
    cuoi = p;
}
```

4. Tạo danh sách (dữ liệu được nhập từ bàn phím)

Code:

```
void taoDS()
{
    char hoten[30];
    float diem;
    dau = NULL; cuoi = NULL;

    do                                      //lặp lại việc nhập 1 nút
    {
        printf("\nNhap ho ten: ");          //đọc "họ tên" từ bàn phím
        fflush(stdin); gets(hoten);
        if (hoten[0])                       //nếu họ tên hợp lệ (khác rỗng)
        {
            printf("Nhap diem: ");          //đọc "điểm" từ bàn phím
            fflush(stdin); scanf("%f", &diem);
            themCuoi(hoten, diem);          //thêm vào cuối danh sách
            //hoặc: themDau(hoten, diem);
        }
    } while (hoten[0]);                     //lặp trong khi hoten khác rỗng
}
```

5. Hiển thị thông tin điểm của tất cả các sinh viên trong danh sách

Code:

```
void hienThiDS()
{
    Nut *p = dau;
    printf("\nDanh sach sinh vien vua nhap:\n");
    while (p != NULL)
    {
        printf("- %s (%g diem)\n", p->HoTen, p->Diem);
        p = p->Tiep;
    }
}
```

6. Đếm số sinh viên thi lại

Code:

```
int soSvThiLai()
{
    int dem = 0;                            //khởi tạo
    Nut *p = dau;
    while (p != NULL)                       //duyệt danh sách
```

```
    {
        if (p->Diem < 5) dem++;
        p = p->Tiep;
    }
    return dem;                             //trả về kết quả
}
```

7. Hiển thị các sinh viên có điểm cao nhất

Code:

```
void hienThiSvDiemMax()
{
    int max = 0;                            //khởi tạo
    Nut *p = dau;                           //duyệt lần 1 để tìm điểm max
    while (p != NULL)
    {
        if (p->Diem > max) max = p->Diem;
        p = p->Tiep;
    }
    printf("\nNhung sinh vien co diem toi da (%g diem):\n");
    p = dau;                                //duyệt lần 2 để hiển thị những sv có điểm max
    while (p != NULL)
    {
        if (p->Diem == max)
            printf("- %s (%g diem)\n", p->HoTen, p->Diem);
        p = p->Tiep;
    }
}
```

8. Tìm kiếm và hiển thị thông tin điểm của các sinh viên theo TÊN

Code:

```
char *trichTen(char *hoten)
{
    int k = strlen(hoten) - 1;
    while (k >= 0 && hoten[k] != ' ') k--;
    return (hoten + k + 1);
}

void timTheoTen(char ten[10])
{
    printf("\nNhung sinh vien co ten '%s':\n", ten);
    Nut *p = dau;
    while (p != NULL)                       //duyệt danh sách
    {
        if (strcmp(trichTen(p->HoTen), ten) == 0)
            printf("- %s (%g diem)\n", p->HoTen, p->Diem);
        p = p->Tiep;
    }
}
```

9. **Hàm `main` để thử nghiệm các hàm vừa xây dựng ở trên**

Code:

```
int main()
{
    taoDS();
    hienThiDS();

    int k = soSvThiLai();
    if (k == 0)
        printf("\nKhong co sinh vien nao phai thi lai.\n");
    else
        printf("\nCo %d sinh vien phai thi lai.\n", k);

    timTheoTen("Huan");
    getch();
    return 0;
}
```

Bài 2. Danh sách SỐ NGUYÊN TĂNG

Sử dụng danh sách liên kết đơn để quản lý dãy các số nguyên tăng dần. Mỗi chút chỉ chứa giá trị là một số nguyên. Xây dựng chương trình gồm các thao tác (hàm) sau:

- Thêm một nút vào cuối danh sách.
- Đọc hai danh sách (A và B) từ tập tin văn bản `SONGUYEN.INP`. Dòng đầu ghi hai số `m` và `n`. Dòng 2 ghi `m` số nguyên của dãy A. Dòng 3 ghi `n` số nguyên của dãy B.
- Kiểm tra xem mỗi danh sách có theo thứ tự tăng dần hay không.
- Thêm một nút vào danh sách (giả sử danh sách đã có thứ tự tăng dần) sao cho sau khi thêm danh sách vẫn đảm bảo có thứ tự tăng dần.
- Ghép 2 danh sách A và B thành danh sách C sao cho danh sách C vẫn đảm bảo thứ tự tăng dần.
- Hiển thị danh sách.

Ví dụ về nội dung tập tin `SONGUYEN.INP`:

```
3 4
4 6 9
2 5 8 10
```

1. **Khai báo danh sách.**

Do chương trình làm việc trên nhiều danh sách nên để tiện và tối ưu ta sử dụng cách 3 trong **bài 1** ở trên, nghĩa là ta tạo thêm kiểu `TroNut`. Cụ thể khai báo như sau:

```
typedef struct SoNguyen
{
    int GiaTri;
    struct SoNguyen *Tiep;
} Nut;
typedef Nut *TroNut;
TroNut dau1, dau2;
```

Ở đây, ta sử dụng hai con trỏ `dau1` và `dau2` để quản lý 2 danh sách A và B. Con trỏ `cuoi` chỉ được sử dụng trong hàm tạo danh sách, do vậy ta sẽ khai báo trong hàm này.

2. **Thêm một nút vào cuối danh sách.**

Cách thêm một nút vào cuối danh sách thì như trình bày ở phần các thao tác cơ bản trên DSLK (mục thêm cuối). Tuy nhiên, khi áp dụng cho trường hợp này thì nhiều bạn sẽ cảm thấy lúng túng vì ở trên chỉ có MỘT danh sách trong khi ở đây chúng ta có đến HAI. Do vậy ta sẽ chỉnh lại một tí để hàm `themCuoi` có thể được sử dụng cho NHIỀU danh sách (chứ không chỉ một hay hai) bằng cách đưa con trỏ đầu và cuối vào tham số của hàm. Có nhiều cách để làm điều này tuy nhiên dưới đây ta sẽ sử dụng phương pháp truyền tham chiếu trong chương trình con, mã nguồn cụ thể như sau:

```
1  void themCuoi(TroNut &dau, TroNut &cuoi, int x)
2  {
3      TroNut p = (TroNut)malloc(sizeof(SoNguyen));
4      p->GiaTri = giatri;
5      p->tiep = NULL;
6      if (dau == NULL)
7          dau = p;
8      else
9          cuoi->tiep = p;
10     cuoi = p;
11 }
```

3. **Đọc hai danh sách (A và B) từ tập tin văn bản `SONGUYEN.INP`.**

Rõ ràng ta cần hai con trỏ để quản lý danh sách A (dau1, cuoi1) và hai con trỏ để quản lý danh sách B (dau2, cuoi2). Ban đầu các con trỏ này phải được khởi tạo NULL (danh sách rỗng). Sau khi mở tập tin bằng lệnh `fopen`, ta đọc hai giá trị m và n (dòng thứ nhất trong tập tin `SONGUYEN.INP`. Để đọc danh sách A từ dòng thứ 2 của tập tin, ta sẽ đọc m số nguyên. Với mỗi số nguyên `x`, ta gọi hàm `themCuoi` với tham số `dau1`, `cuoi1` và `x`. Hoàn toàn tương tự với danh sách B.

```
void docDS()
{
    TroNut cuoi1 = NULL, cuoi2 = NULL;
    dau1 = NULL; dau2 = NULL;
    int i, n, m, x;
    FILE *f = fopen("SONGUYEN.INP", "r");
    fscanf(f, "%d %d", &m, &n);
    for (i = 0; i < m; i++)
    {
        fscanf(f, "%d", &x);
        themCuoi(dau1, cuoi1, x);
    }
    for (i = 0; i < n; i++)
    {
        fscanf(f, "%d", &x);
        themCuoi(dau2, cuoi2, x);
    }
    fclose(f);
}
```

4. **Kiểm tra xem mỗi danh sách có theo thứ tự tăng dần hay không.**

Ý tưởng: Để kiểm tra xem danh sách có theo thứ tự tăng dần hay không, ta sẽ cố gắng tìm vị trí mà giá trị tại nút này lớn hơn giá trị tại nút tiếp theo. Nếu tìm thấy thì chứng tỏ danh sách danh sách không theo thứ tự tăng dần.

Cài đặt: Để "tìm" nút mong muốn (giá trị lớn hơn nút tiếp theo) rõ ràng ta cần phải lặp. Do chưa biết trước số lần lặp nên sử dụng vòng lặp `while`. Mã cụ thể như dưới đây:.

```
int dayTang(TroNut dau)
{
    if (dau == NULL) return 1;      //dãy rỗng coi như là "dãy tăng dần"
    TroNut p = dau;
    while (p->Tiep != NULL && p->GiaTri <= p->Tiep->GiaTri)
        p = p->Tiep;
    if (p->Tiep == NULL) return 1;   //nếu tìm không thấy (duyệt hết dãy) thì chắc chắn dãy là "tăn
    return 0; //nếu duyệt KHÔNG hết dãy (tìm thấy) thì chắc chắn dãy là "không tăng dần"
}
```

5. **Thêm một nút vào danh sách (giả sử danh sách đã có thứ tự tăng dần) sao cho sau khi thêm danh sách vẫn đảm bảo có thứ tự tăng dần.**

Để làm điều này, ta cần tìm vị trí mà giá trị tại nút này không nhỏ hơn giá trị tại nút tiếp theo. Ta sẽ thực hiện vòng lặp tương tự câu trên (kiểm tra tính tăng dần của danh sách). Sau khi tìm được ta sẽ đưa nút mới (p) vào ngay sau nút này (q). Có một trường hợp đặc biệt cần chú ý là danh sách rỗng hoặc nút

đầu tiên của danh sách có giá trị lớn hơn giá trị muốn thêm vào. Trường hợp này ta chỉ việc thực hiện thao tác "thêm đầu" (xem phần cơ bản, mục thêm đầu). Mã nguồn đầy đủ như sau:

```
void themPhanTu(TroNut &dau, int giatri)
{
    TroNut p = (TroNut)malloc(sizeof(Nut));
    p->GiaTri = giatri;

    if (dau == NULL || (dau != NULL && dau->GiaTri > giatri))
    {
        p->Tiep = dau;
        dau = p;
    }
    else
    {
        TroNut q = dau;
        while (q->Tiep != NULL && q->Tiep->GiaTri < giatri)
            q = q->Tiep;
        p->Tiep = q->Tiep;
        q->Tiep = p;
    }
}
```

6. Hợp 2 danh sách A và B thành danh sách C sao cho danh sách C vẫn đảm bảo thứ tự tăng dần.

Ta sẽ thực hiện vòng lặp để lần lượt đọc từng nút từ một trong hai danh sách A và B để thêm vào danh sách C sao cho đúng thứ tự. Ban đầu ta sẽ xuất phát từ đầu của 2 danh sách, tại mỗi bước lặp ta sẽ chỉ đưa MỘT nút từ danh sách A hoặc danh sách B sang danh sách C. Ta sẽ đưa nút từ danh sách A nếu:

1. Danh sách B đã hết (`dau2=NULL`) hoặc
2. Giá trị nút hiện tại của danh sách B lớn hơn diá trị nút hiện tại của danh sách A (`dau2->GiaTri > dau1->GiaTri`)

Ngược lại, ta sẽ đưa nút từ danh sách B. Hàm `themDau` sẽ được sử dụng để đưa nút từ danh sách (A hoặc B) sang danh sách C. Con trỏ `dau` quản lý danh sách C sẽ là giá trị trả về của hàm.

```
TroNut ghepDS(TroNut dau1, TroNut dau2)
{
    TroNut dau = NULL, cuoi = NULL;
    int giatri;
    while (dau1 != NULL || dau2 != NULL)
    {
        if (dau2 == NULL || (dau1 != NULL && dau2 != NULL && dau2->GiaTri > dau1->GiaTri))
        {
            giatri = dau1->GiaTri;
```

```
            dau1 = dau1->Tiep;
        }
        else
        {
            giatri = dau2->GiaTri;
            dau2 = dau2->Tiep;
        }
        themCuoi(dau, cuoi, giatri);
    }
    return dau;
}
```

7. Hiển thị nội dung toàn bộ danh sách.

```
void hienThi(TroNut dau)
{
    TroNut p = dau;
    while (p != NULL)
    {
        printf("%d ", p->GiaTri);
        p = p->tiep;
    }
    printf("\n");
}
```

8. Chương trình hoàn thiện

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

typedef struct SoNguyen
{
    int GiaTri;
    struct SoNguyen *Tiep;
} Nut;
typedef Nut *TroNut;
TroNut dau1, dau2;

void themCuoi(TroNut &dau, TroNut &cuoi, int giatri)
{
    TroNut p = (TroNut)malloc(sizeof(Nut));
    p->GiaTri = giatri;
    p->Tiep = NULL;
    if (dau == NULL)
        dau = p;
    else
        cuoi->Tiep = p;
    cuoi = p;
}

void docDS()
{
```

```
    TroNut cuoi1 = NULL, cuoi2 = NULL;
    dau1 = NULL; dau2 = NULL;
    int i, n, m, x;
    FILE *f = fopen("SONGUYEN.INP", "r");
    fscanf(f, "%d %d", &m, &n);                 //đọc giá trị m và n
    for (i = 0; i < m; i++)                      //đọc m giá trị của mảng a và tạo danh sách 1
    {
        fscanf(f, "%d", &x);
        themCuoi(dau1, cuoi1, x);
    }
    for (i = 0; i < n; i++)                      //đọc n giá trị của mảng b và tạo danh sách 2
    {
        fscanf(f, "%d", &x);
        themCuoi(dau2, cuoi2, x);
    }
    fclose(f);
}

void hienThi(TroNut dau)
{
    TroNut p = dau;
    while (p != NULL)
    {
        printf("%d ", p->GiaTri);
        p = p->Tiep;
    }
    printf("\n");
}

int dayTang(TroNut dau)
{
    if (dau == NULL) return 1;      //dãy rỗng coi như là "dãy tăng dần"
    TroNut p = dau;
    while (p->Tiep != NULL && p->GiaTri < p->Tiep->GiaTri)
        p = p->Tiep;
    if (p->Tiep == NULL) return 1;   //nếu duyệt hết dãy thì chắc chắn dãy là "tăng dần"
    return 0; //nếu duyệt KHÔNG hết dãy (dừng ở đâu đó) thì chắc chắn dãy là "không tăng dần"
}

void themPhanTu(TroNut &dau, int giatri)
{
    TroNut p = (TroNut)malloc(sizeof(Nut));
    p->GiaTri = giatri;

    if (dau == NULL || (dau != NULL && dau->GiaTri > giatri))
    {
        p->Tiep = dau;
        dau = p;
    }
    else
    {
        TroNut q = dau;
        while (q->Tiep != NULL && q->Tiep->GiaTri < giatri)
            q = q->Tiep;
        p->Tiep = q->Tiep;
        q->Tiep = p;
```

```
    }
}

TroNut ghepDS(TroNut dau1, TroNut dau2)
{
    TroNut dau = NULL, cuoi = NULL;
    int giatri;
    while (dau1 != NULL || dau2 != NULL)
    {
        if (dau2 == NULL || (dau1 != NULL && dau2 != NULL && dau2->GiaTri > dau1->GiaTri))
        {
            giatri = dau1->GiaTri;
            dau1 = dau1->Tiep;
        }
        else
        {
            giatri = dau2->GiaTri;
            dau2 = dau2->Tiep;
        }
        themCuoi(dau, cuoi, giatri);
    }
    return dau;
}

int main()
{
    docDS();
    printf("Danh sach 1: ");
    hienThi(dau1);
    printf("Danh sach 2: ");
    hienThi(dau2);

    themPhanTu(dau1, 7);
    printf("Danh sach 1 sau khi them phan tu gia tri 7: ");
    hienThi(dau1);

    themPhanTu(dau2, 1);
    printf("Danh sach 2 sau khi them phan tu gia tri 1: ");
    hienThi(dau2);

    printf("Danh sach sau khi ghep: ");
    hienThi(ghepDS(dau1, dau2));
    getch();
}
```

Kết quả thực thi chương trình (màn hình console) sẽ như sau:

```
Danh sach 1: 4 6 9
Danh sach 2: 2 5 8 10
Danh sach 1 sau khi them phan tu gia tri 7: 4 6 7 9
Danh sach 2 sau khi them phan tu gia tri 1: 1 2 5 8 10
```

```
Danh sach sau khi ghep: 1 2 4 5 6 7 8 9 10
```

Viết Bình luận

Bình luận (9)

Le Duc Trieu

Cảm ơn !
[19/03/2015 20:36]

cảm ơn AD rất nhiều - pro !

TPN

Thêm nút vào giữa
[02/03/2015 11:10]

Ai có thể giúp mình thêm nút vào giữa ko nhỉ

Trung

Excellent!!
[03/02/2015 14:07]

Bài viết rất chi tiết và cặn kẽ, có đâu tư rất công phu

tuha

thank
[29/01/2015 22:52]

Cảm ơn nhiều lắm ạ!

nguyên minh ngoc

bài viết hay
[30/10/2014 01:33]

cảm ơn ad
mong ad có nhiều bài viết trên các chương trinh khác nhau như Java,c#, và trên c có nhiều bài nữa

quang vu

bài viết rất hay
[27/09/2014 23:39]

bài viết rất hay và bổ ích, trình bày bố cục rất trực quan, dễ hiểu. Mình đang rất cần tài liệu này, cảm ơn tác giả !