Trường Đại học Khoa học Tự nhiên
Khoa Công nghệ thông tin
Bộ môn Tin học cơ sở

NHẬP MÔN LẬP TRÌNH

Đặng Bình Phương
dbphuong@fit.hcmuns.edu.vn

HÀM NÂNG CAO (PHẦN 2)

Nội dung

1. Tham số …
2. Khuôn mẫu hàm
3. Nạp chồng hàm
4. Nạp chồng toán tử

Tham số ...

- Khai báo

```
<kiểu trả về> <tên hàm>(<dsts biết trước>, …)
{
    …
}
```

- Ý nghĩa
  - Hàm có số lượng tham số không biết trước và thường cùng kiểu (không được là char, unsigned char, float).
  - Phải có ít nhất 1 tham số biết trước.
  - Tham số … đặt ở cuối cùng.

Tham số ...

- Ví dụ

```
void XuatTong1(char *msg, int n, …)
{
    // Các lệnh ở đây
}

void XuatTong2(char *msg, …)
{
    // Các lệnh ở đây
}

int Tong(int a, …)
{
    // Các lệnh ở đây
}
```

Truy xuất danh sách tham số …

- Sử dụng kiểu và các macro sau (stdarg.h)
  - va_list : kiểu dữ liệu chứa các tham số có trong …
  - va_start(va_list ap, lastfix) : macro thiết lập ap chỉ đến tham số đầu tiên trong … với lastfix là tên tham số cố định cuối cùng.
  - type va_arg(va_list ap, type) : macro trả về tham số có kiểu type tiếp theo.
  - va_end(va_list ap) : macro giúp cho hàm trả về giá trị một cách "bình thường".

Tham số ...

- Ví dụ

```
#include <stdarg.h>
void XuatTong1(char *msg, int n, …)
{
    va_list ap;
    va_start(ap, n); // ts cố định cuối cùng
    int value, s = 0;
    for (int i=0; i<n; i++)
    {
        value = va_arg(ap, int);
        s = s + value;
    }
    va_end(ap);
    printf("%s %d", msg, s);
}
```

Tham số ...

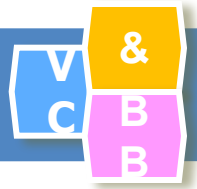
- Ví dụ

```
#include <stdarg.h>
void XuatTong2(char *msg, …)
{
    va_list ap;
    va_start(ap, msg); // ts cố định cuối
    int value, s = 0;
    while ((value = va_arg(ap, int)) != 0)
    {
        s = s + value;
    }
    va_end(ap);
    printf("%s %d", msg, s);
}
```

Tham số ...

- Ví dụ

```
#include <stdarg.h>
int Tong(int a, …)
{
    va_list ap;
    va_start(ap, n); // ts cố định cuối cùng
    int value, s = a;
    while ((value = va_arg(ap, int)) != 0)
    {
        s = s + value;
    }
    va_end(ap);
    return s;
}
```



Khuôn mẫu hàm

- Viết hàm tìm số nhỏ nhất trong 2 số
  - Viết các hàm khác nhau để tìm min 2 số int, 2 số long, 2 số float, 2 số double, 2 phân số…
- Nhược điểm
  - Hàm bản chất giống nhau nhưng khác kiểu dữ liệu nên phải viết nhiều hàm giống nhau.
  - Sửa 1 hàm phải sửa những hàm còn lại.
  - Không thể viết đủ các hàm cho mọi trường hợp do còn nhiều kiểu dữ liệu khác.

Khuôn mẫu hàm

- Khái niệm
  - Viết một hàm duy nhất nhưng có thể sử dụng cho nhiều kiểu dữ liệu khác nhau.
- Cú pháp
  - tempate <ds mẫu tham số> <khai báo hàm>
- Ví dụ

```
template <class T> <khai báo hàm>

hoặc

template <class T1, class T2> <khai báo hàm>
```

Khuôn mẫu hàm

- Ví dụ

```
template <class T>
T min(T a, T b)
{
    if (a < b)
        return a;
    return b;
}

void main()
{
    int a = 2912, b = 1706;
    int m = min<int>(a, b);
    printf("So nho nhat la %d", m);
}
```

Khuôn mẫu hàm

- Lợi ích của việc sử dụng khuôn mẫu hàm
  - Dễ viết, do chỉ cần viết hàm tổng quá nhất.
  - Dễ hiểu, do chỉ quan tâm đến kiểu tổng quát nhất.
  - Có kiểu an toàn do trình biên dịch kiểm tra kiểu lúc biên dịch chương trình.
  - Khi phối hợp với sự quá tải hàm, quá tải toán tử hoặc con trỏ hàm ta có thể viết được các chương trình rất hay, ngắn gọn, linh động và có tính tiến hóa cao.

Nạp chồng hàm

- Nhu cầu
  - Thực hiện một công việc với nhiều cách khác nhau. Nếu các hàm khác tên sẽ khó quản lý.
- Khái niệm nạp chồng/quá tải (overload) hàm
  - Hàm cùng tên nhưng có tham số đầu vào hoặc đầu ra khác nhau.
  - Nguyên mẫu hàm (prototype) khi bỏ tên tham số phải khác nhau.
  - Cho phép người dùng chọn phương pháp thuận lợi nhất để thực hiện công việc.

Nạp chồng hàm

- Ví dụ
  - Nhập mảng theo nhiều cách

```
void Nhap(int a[], int &n)
{
    // Nhập n rồi nhập mảng a
}
void Nhap(int a[], int n)
{
    // Nhập mảng a theo n truyền vào
}
int Nhap(int a[])
{
    // Nhập n, nhập mảng a rồi trả n về
}
```

Nạp chồng hàm

- Ví dụ
  - Gán giá trị cho PHANSO

```
void Gan(PHANSO &ps, int tu, int mau)
{
   ps.tu = tu;
   ps.mau = mau;
}

void Gan(PHANSO &ps, int tu)
{
   ps.tu = tu;
   ps.mau = 1;
}
```

Nạp chồng hàm

- Chú ý
  - Các hàm sau đây là như nhau

```
int Tong(int a, int b)  // int Tong(int, int)
{
   return a + b;
}
int Tong(int b, int a)  // int Tong(int, int)
{
   return a + b;
}
int Tong(int x, int y)  // int Tong(int, int)
{
   return x + y;
}
```

Nạp chồng hàm

- Sự nhập nhằng, mơ hồ (ambiguity)
  - Do sự tự chuyển đổi kiểu

```
float f(float x) { return x / 2; }
double f(double x) { return x / 2; }

void main()
{
    float x = 29.12;
    double y = 17.06;
    printf("%.2f\n", f(x)); // float
    printf("%.21f\n", f(y));// double
    printf("%.2f", f(10));  // ???
}
```

Nạp chồng hàm

- Sự nhập nhằng, mơ hồ (ambiguity)
  - Do sự tự chuyển đổi kiểu

```
void f(unsigned char c)
{
   printf("%d", c);
}
void f(char c)
{
   printf("%c", c);
}
void main()
{
   f('A');  // char
   f(65);// ???
}
```

Nạp chồng hàm

- Sự nhập nhằng, mơ hồ (ambiguity)
  - Do việc sử dụng tham chiếu

```
int f(int a, int b)
{
   return a + b;
}
int f(int a, int &b)
{
   return a + b;
}
void main()
{
   int x = 1, y = 2;
   printf("%d", f(x, y));  // ???
}
```

Nạp chồng hàm

- Sự nhập nhằng, mơ hồ (ambiguity)
  - Do việc sử dụng tham số mặc định

```
int f(int a)
{
   return a*a;
}
int f(int a, int b = 0)
{
   return a*b;
}
void main()
{
   printf("%d\n", f(2912, 1706));
   printf("%d\n", f(2912));//???
}
```

Nạp chồng toán tử

- Khái niệm
  - Giống như quá tải hàm.
  - Có một số quy định khác về tham số.
  - Tạo thêm hàm toán tử (operator function) để nạp chồng toán tử.

```
<kiểu trả về> operator#(<ds tham số>)
{
   // Các thao tác cần thực hiện
}
```

  - # là toán tử (trừ . :: .* ?), <ds tham số> phụ thuộc vào toán tử được nạp chồng.

Nạp chồng toán tử

- Toán tử hai ngôi
  - Toán tử gán (=), số học (+, –, *, /, %), quan hệ (<, <=, >, >=, !=, ==), luận lý (&&, ||, !)
  - Gồm có hai toán hạng
  - Có thể thay đổi toán hạng về trái
  - Có thể trả kết quả về cho phép toán tiếp theo
- Toán tử một ngôi
  - Toán tử tăng giảm (++, – –), toán tử đảo dấu (–)
  - Chỉ có một toán hạng

Nạp chồng toán tử

- Toán tử hai ngôi
  - Toán tử + (cho hai PHANSO)

```
typedef struct {int tu, mau;} PHANSO;

PHANSO operator+(PHANSO ps1, PHANSO ps2)
{
    PHANSO ps;
    ps.tu = ps1.tu*ps2.mau + ps2.tu*ps1.mau;
    ps.mau = ps1.mau*ps2.mau;
    return ps;
}
…
PHANSO a = {1, 2}, b = {3, 4}, c = {5, 6};
PHANSO d = a + b + c;// ⇔ d = a + (b + c)
```

Nạp chồng toán tử

- Toán tử hai ngôi
  - Toán tử + (cho hai PHANSO)

```
typedef struct {int tu, mau;} PHANSO;

void operator+(PHANSO &ps1, PHANSO ps2)
{
    PHANSO ps;
    ps.tu = ps1.tu*ps2.mau + ps2.tu*ps1.mau;
    ps.mau = ps1.mau*ps2.mau;
    ps1 = ps;
}
…
PHANSO a = {1, 2}, b = {3, 4}, c = {5, 6};
PHANSO d = a + b + c;// Lỗi
```

Nạp chồng toán tử

- Toán tử hai ngôi
  - Toán tử + (cho hai PHANSO)

```
typedef struct {int tu, mau;} PHANSO;

PHANSO operator+(PHANSO ps1, int n)
{
    PHANSO ps;
    ps.tu = ps1.tu + ps1.mau*n;
    ps.mau = ps1.mau;
    return ps;
}
…
PHANSO a = {1, 2};
PHANSO b = a + 2; // OK
PHANSO c = 2 + a; // Lỗi sai thứ tự toán hạng
```

Nạp chồng toán tử

- Toán tử hai ngôi
  - Toán tử == (cho hai PHANSO)

```
typedef struct {int tu, mau;} PHANSO;

int operator==(PHANSO ps1, PHANSO ps2)
{
   if (ps1.tu*ps2.mau == ps2.tu*ps1.mau)
      return 1;
   return 0;
}
…
PHANSO a = {1, 2}, b = {2, 4};
if (a == b)
   printf("a bằng b");
```

Nạp chồng toán tử

- Toán tử một ngôi
  - Toán tử tăng ++ (trước)

```
typedef struct {int tu, mau;} PHANSO;

PHANSO operator++(PHANSO &ps)
{
   ps.tu = ps.tu + ps.mau;
   return ps;
}
…
PHANSO a1 = {1, 2}, a2 = {1, 2};
PHANSO c1 = ++a1;
PHANSO c2 = a2++; // OK nhưng warning ++ sau
```

Nạp chồng toán tử

- Toán tử một ngôi
  - Toán tử tăng ++ (sau)

```
typedef struct {int tu, mau;} PHANSO;

PHANSO operator++(PHANSO &ps, int notused)
{
   ps.tu = ps.tu + ps.mau;
   return ps;
}
…
PHANSO a = {1, 2}, b = {3, 4};
PHANSO c1 = ++a;  // operator++(ps)
PHANSO c2 = a++;  // operator++(ps, 0)
```

Nạp chồng toán tử

- Toán tử một ngôi
  - Toán tử đảo dấu –

```
typedef struct {int tu, mau;} PHANSO;

PHANSO operator–(PHANSO ps)
{
   ps.tu = –ps.tu;
   return ps;
}
…
PHANSO a = {1, 2};
PHANSO b = –a;
```

Bài tập

- Bài 1: Viết chương trình tính tổng các số nguyên truyền vào hàm (có truyền thêm số lượng).
- Bài 2: Sửa lại bài 1 để cho phép người dùng tính tổng các số có kiểu bất kỳ được truyền vào hàm (có truyền thêm số lượng)
- Bài 3: Viết chương trình sắp xếp mảng tăng dần. Các phần tử của mảng có kiểu bất kỳ (char, int, long, float, double, phân số, sinh viên …)
- Bài 4: Sửa lại bài 3 để cho phép người dùng thay đổi quy luật sắp xếp (tăng, giảm, âm tăng dương giảm, …)