



Function

Xây dựng hàm, chương trình con

Môn học: Cơ sở lập trình [*Buổi 6-7*]

GV: Nguyễn Mai Huy

Sự cần thiết của hàm

- Khi độ phức tạp của chương trình tăng lên, việc sử dụng quá nhiều mã lệnh trong 1 chương trình mang lại nhiều nhược điểm:
 - **Rối rắm, Khó kiểm soát**
 - **Khó chỉnh sửa** (*Khi bảo trì, nâng cấp phần mềm*)
 - **Có những công việc bị lặp lại một cách không cần thiết ...**
- Hàm là một kỹ thuật khá phổ biến trong các ngôn ngữ lập trình bậc cao dùng để cho phép người lập trình tự tạo ra các công cụ để sử dụng trong chương trình của mình
- Việc tổ chức chương trình với các hàm thành phần đem lại một số ưu điểm như sau
 - **Kiểm soát mã nguồn khoa học**: Việc phân tách chương trình thành những đơn vị nhỏ hơn, độc lập với nhau do đó việc quản lý mã nguồn chương trình được chặt chẽ hơn, thuận lợi cho việc kiểm soát và phát hiện lỗi, dễ mở rộng và bảo trì trong tương lai
 - **Giảm kích thước** (*Khối lượng mã lệnh*) cho chương trình do khi thực hiện nhiệm vụ nào đó, chỉ cần 1 lời gọi hàm ngắn gọn tại nơi cần thực hiện
 - **Nhất quán về mặt giải thuật** (*Khi cần chỉnh sửa, chỉ cần điều chỉnh trong nội dung của hàm là tất cả các vị trí gọi hàm trong chương trình đều sẽ chịu ảnh hưởng*)

Khai báo và sử dụng hàm

Cú pháp

```
<kiểu_DL> <tên_hàm>([tham_số], [tham_số], ...)  
{  
    ...  
    ...  
    return <giá_trị>;  
}
```



Thân hàm

- **<kiểu_DL>**: Có thể là 1 kiểu dữ liệu cơ bản (*byte, short, int, long, float, ...*) hoặc một kiểu cấu trúc do người dùng tự định nghĩa (*sinhVien, hangHoa, ...*) dĩ nhiên kiểu dữ liệu được khai báo cho hàm phải phù hợp với giá trị mà nó sẽ trả về cho nơi gọi
- **<tên_hàm>**: là chuỗi ký tự do người lập trình tự quyết định, tuy nhiên tên hàm phải đặt theo quy ước về đặt tên cho danh định trong chương trình. Tức là luôn phải bắt đầu bởi ký tự, chỉ sử dụng chữ cái hoặc chữ số, không được sử dụng ký tự đặc biệt và dĩ nhiên, tên hàm phải là duy nhất, không được trùng với tên của 1 hàm đã có
- **[tham_số]** thường được dùng để mô tả các giá trị (*hoặc các biến*) sẽ truyền vào cho hàm tại thời điểm gọi hàm. Có thể nói “*tham số được xem như nguyên liệu để cung cấp cho quá trình xử lý của hàm*”. Tham số thường bao gồm 2 phần:
<kiểu_DL> <tên_tham_số>

Nguyễn Mai Huy - nmai.huy@boduca.com

Ví dụ

- Xây dựng hàm tính n giai thừa

long giaiThua(**int** n)

```
{  
    long kq = 1;  
    for(int i, i<=n;i++)  
        kq *=i;  
    return kq;  
}
```

- Viết các hàm phục vụ cho mục đích sau
 - Tính diện tích hình tròn
 - Tính chu vi hình tròn

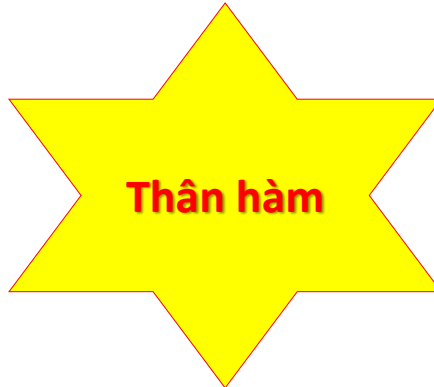
Hàm không trả về giá trị

- Trong trường hợp cần xây dựng hàm để thực hiện cho một mục tiêu nào đó trong chương trình; tuy nhiên, sau khi thi hành, chương trình không cần nhận giá trị trả về của hàm, ta có thể thay thế <kiểu_DL> bằng khai báo void

- Cú pháp

void <tên_hàm>([tham_số], [tham_số], ...)

```
{  
    ...  
    ...  
    return;  
}
```



Ví dụ

- Xây dựng hàm phục vụ cho mục đích in ra bảng cửu chương thứ n

void bangCuuChuong(int n)

{

for(int i=1; i<=10;i++)

printf(“ %2d x %2d = %3d \n”, n, i, n*i);

}

- Tương tự, xây dựng hàm phục vụ cho các yêu cầu sau:
 - Vẽ ra màn hình hình chữ nhật đặc có kích thước d x r với ký tự vẽ là “*” và d, r là các tham số truyền cho hàm
 - Xây dựng hàm phục vụ cho iệc đổi số từ thập phân ra nhị phân, tham số truyền cho hàm là 1 số nguyên dương, kết quả trả về là chuỗi chỉ bao gồm 2 ký số 0 và 1 tượng trưng cho giá trị trong hệ nhị phân cần đổi

Khai báo hàm trong chương trình

```
import <thư_viện>

public class <tenChươngTrình>{
    <kiểu_DL> <tên_hàm>([Tham_số], [Tham_số], ...)
    {
        ... ///--- Thân hàm
        ...
        return <giá_trị>;
    }
    ..... ///--- Các khai báo hàm khác trong chương trình
    public static void main(String args[])
    {
        ... ///--- Chương trình chính
        ...
        ... ///--- Gọi hàm trong chương trình chính
        ...
    }
}
```

Hình thức thứ nhất
Khai báo và định nghĩa
hàm trước vị trí của hàm
main

“Lời gọi hàm” trong chương trình

Sau khi đã khai báo và định nghĩa hàm, trong chương trình chính và các hàm khác, ta có thể gọi sử dụng hàm để phục vụ cho mục tiêu xử lý dữ liệu. Một “**lời gọi hàm**” như vậy có thể được thực hiện như là 1 lệnh độc lập trong chương trình, hoặc cũng có thể sử dụng chúng như là 1 thành phần của 1 biểu thức. Cú pháp:

<tên_hàm>([tham_số], ...);

Hoặc

<kiểu_DL> <biến> = <tên_hàm>([tham_số], ...);

VD:

- Gọi hàm vẽ hình chữ nhật với dài x rộng: 20, 5

veHCN(20, 5);

- Gọi hàm tính giai thừa của 8

long gt = giaiThua(8);

Các hình thức truyền tham số

- Như đã đề cập trong phần trước, tham số của hàm được xem như là “nguyên liệu” để phục vụ cho quá trình xử lý của hàm thường được mô tả trong cặp dấu ngoặc “()”, có hai hình thức truyền tham số cho hàm:
 - **Tham trị**: nghĩa là các tham số truyền cho hàm không bị thay đổi giá trị sau khi hàm thi hành. Mặc nhiên, các tham số của hàm đều được xem là “*tham trị*”
 - **Tham chiếu**: Trong những tình huống nhất định, người lập trình mong muốn, sau khi hàm xử lý thì các tham số được truyền cho hàm dưới dạng các biến có thể sẽ bị thay đổi giá trị ở nơi thực hiện “lời gọi hàm”. Trong trường hợp này, tham số phải là đối tượng của một class. Kỹ thuật tham chiếu trong java, không áp dụng cho các biến thuộc loại **primitive data types**

Ví dụ minh họa

truyền tham số dưới dạng tham chiếu

```
public class Example {  
    int minhHoa(int &a, int &b)  
    {  
        int kq = a * b;  
        a *=2;  
        b++;  
        return kq;  
    }  
    public static void main(String args[])  
    {  
        int x = 7, y=10;  
        int tich = minhHoa(x, y);  
        printf("Sau khi gọi hàm: x=%d – y=%d ", x,y);  
        printf("\n Kết quả của hàm tich= %d", tich);  
        getch();  
    }  
}
```

Tâm vực của biến

- Tùy vào vị trí khai báo mà các biến trong chương trình có phạm vi nhận biết và sử dụng khác nhau. Có hai hình thức khai báo biến mà tầm ảnh hưởng (*hay khu vực có thể nhận biết*) thường được đề cập đến trong chương trình
 - **Biến toàn cục**: là những biến nằm ngoài các hàm (*kể cả hàm main trong chương trình*), thường đặt trong khu vực “*đầu tiên, ngay sau khai báo **class***” của chương trình, các biến thuộc loại này được cấp phát bộ nhớ và tồn tại trong suốt thời gian mà chương trình hoạt động. Biến toàn cục có thể được truy xuất và sử dụng ở bất cứ nơi nào trong chương trình (*trong các hàm con và hàm main đều có thể truy xuất, sử dụng các biến thuộc loại này*)
 - **Biến cục bộ**: là những biến được khai báo bên trong các hàm (*hoặc trong khối lệnh*), các biến này chỉ được cấp phát bộ nhớ khi trình biên dịch thực thi hàm (*hay khối lệnh tương ứng*) và khả năng nhận biết cũng như sử dụng của chúng cũng chỉ trong phạm vi của hàm (*hay khối lệnh*) tính từ vị trí khai báo cho đến khi gặp dấu “*}*” là vị trí kết thúc hàm (*hay khối lệnh*) mà thôi

* Cho 1 ví dụ sử dụng biến toàn cục, cục bộ trên bảng

- Đệ quy là 1 khái niệm thường dùng trong toán học để chỉ việc định nghĩa một biểu thức mới dựa trên chính nó nhưng với giá trị thay đổi. Hãy xét tình huống tính tổng các số từ 1 đến n bằng phương pháp quy nạp
- Ta có: $s(n) = 1 + 2 + 3 + \dots + n$
và công thức trên có thể biểu diễn ở dạng sau
 $s(n) = s(n-1) + n$
- $s(n-1) = s(n-2) + n-1$
.
.
.
 $s(2) = s(1) + 2$
 $s(1) = 1$
- Như vậy ta thấy: xuất phát điểm ta có s(1), sau khi có s(1), ta thay giá trị của nó vào để tính được s(2), khi có giá trị của s(2), ta lại sử dụng để tính được s(3), ... cứ thế cho đến khi tính được s(n-1) và kết thúc ta được kết quả s(n) suy diễn lần lượt từ các s(n-1), s(n-2), ..., 1

Hàm đệ quy - Function recursion

- Trong ngôn ngữ C, cơ chế đệ quy được trang bị cho các hàm được định nghĩa trong chương trình để cho phép 1 hàm có thể gọi lại chính nó một cách gián tiếp (*hay trực tiếp*)
- VD. Viết hàm đệ quy để phục vụ cho việc tính giai thừa của n

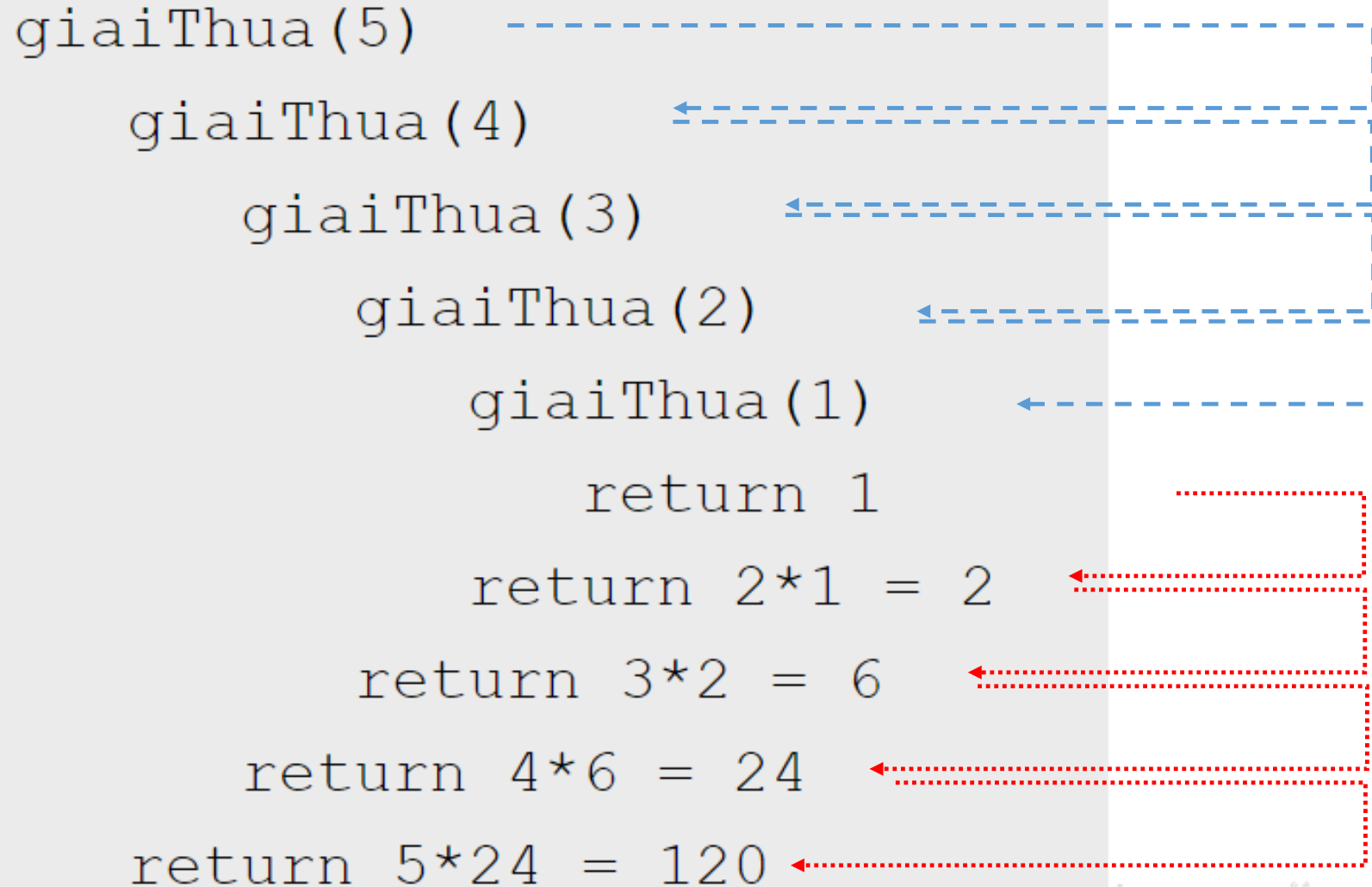
```
public class Example {  
    long giaiThua(int n)  
    {  
        if (n<=1)  
            return 1;  
        else  
            return giaiThua(n-1)*n;  
    }  
    public static void main(String args[])  
    {  
        int x;  
        printf("Muốn tính giai thừa của:"); scanf("%d",&x);  
        long gt = giaiThua(x);  
        printf("%d giai thừa = %l", x, gt);  
        getch();  
    }  
}
```

Cơ chế thực thi hàm trong chương trình

- Khi thi hành chương trình, trình biên dịch gặp một lời gọi hàm, nó sẽ tạm ngừng thực thi các lệnh ở phía sau khối lệnh có lời gọi hàm, đồng thời bắt đầu thực hiện thực thi hàm được gọi theo trình tự sau:
 - Cấp phát bộ nhớ cho các biến cục bộ đã được khai báo trong hàm.
 - Thiết lập giá trị của các tham số truyền vào (tại vị trí có lời gọi hàm) cho các tham số đã được định nghĩa cho hàm tương ứng.
 - Thi hành các lệnh trong thân hàm cho đến khi gặp lệnh **return** hoặc dấu “}” (*kết thúc khối lệnh của hàm*), lúc này các tham số, các biến cục bộ đã cấp phát cho hàm sẽ bị loại bỏ khỏi bộ nhớ đồng thời trở về vị trí ngay sau lời gọi hàm đã thực hiện trước đó để tiếp tục thi hành các lệnh đang tạm ngừng.
- Nếu hàm kết thúc bởi lệnh **return** có chứa giá trị thì giá trị tương ứng sẽ được gán làm kết quả cho hàm. Giá trị của hàm sẽ được sử dụng để tham gia tính toán trong các biểu thức có chứa nó.

Hàm **giaiThua** hoạt động với $n=5$

```
giaiThua(5)  -----  
    giaiThua(4)  -----  
        giaiThua(3)  -----  
            giaiThua(2)  -----  
                giaiThua(1)  -----  
                    return 1  
                return 2*1 = 2  
            return 3*2 = 6  
        return 4*6 = 24  
    return 5*24 = 120
```



luy - nmai huy@bodu a.com

Nhớ gì ?!!!

- Hàm là gì, tại sao phải cần xây dựng hàm. Các lợi ích của việc sử dụng hàm trong chương trình
- Cú pháp khai báo, định nghĩa một hàm trong chương trình
- Có bao nhiêu loại hàm có thể định nghĩa trong chương trình ?
- Các hình thức truyền tham số
- Hiểu thế nào về tầm vực của biến ?
- Vấn đề đệ quy trong chương trình

- **Recursion**

(<https://introcs.cs.princeton.edu/java/23recursion/>)

- **Recursion in Java**

(<https://www.javatpoint.com/recursion-in-java>)