# CƠ SỞ LẬP TRÌNH

## CHƯƠNG TRÌNH CON

# Nội dung

- Khái niệm
- Cách xây dựng hàm
- Tầm tác dụng của biến
- Truyền tham số cho hàm
- Hàm đệ quy
- Một số hàm thông dụng

# 4.1 Khái niệm

- Chương trình con là:
  - Một đoạn chương trình có tên, đầu vào và đầu ra.
  - Có chức năng giải quyết một số vấn đề chuyên biệt cho chương trình chính.
  - Được gọi nhiều lần với các tham số khác nhau.
  - Được sử dụng khi có nhu cầu:
    - Tái sử dụng: có một số chương trình được thực hiện ở nhiều nơi, bản chất không đổi nhưng giá trị các tham số cung cấp khác nhau.
    - Chia để trị: chia chương trình lớn thành các chương trình nhỏ rồi ghép lại.

  ➔ Giúp chương trình trong sáng, dễ hiểu, dễ phát hiện lỗi và cải tiến.

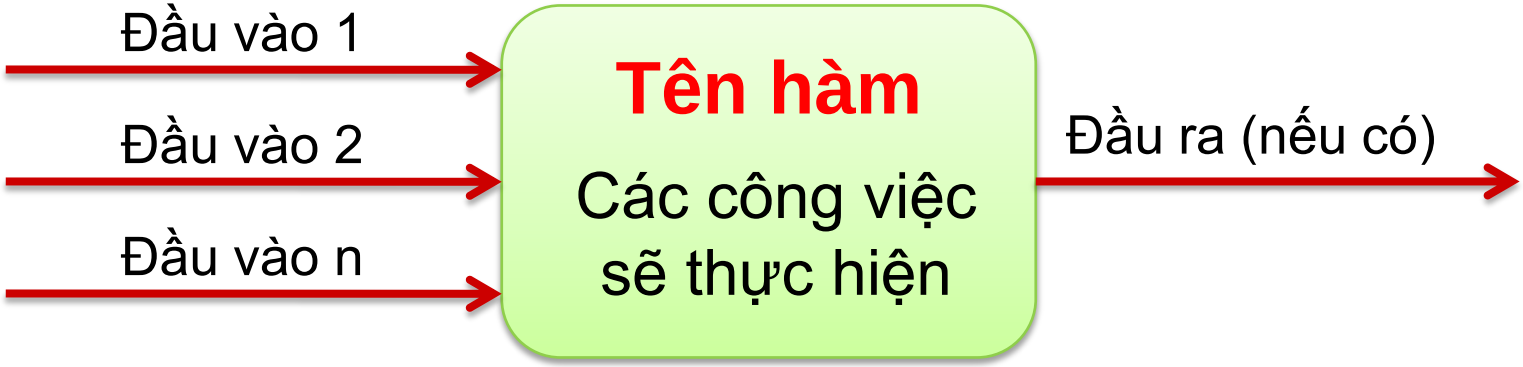
# 4.1 Khái niệm (tt)

- Trong các ngôn ngữ khác, có 2 loại chương trình con:
  - Hàm (function): trả về giá trị thông qua tên hàm, sử dụng trong các biểu thức và không được gọi như một lệnh.
  - Thủ tục: không có giá trị trả về, có thể tồn tại độc lập và được gọi như là một câu lệnh.
- Trong C: **chỉ tồn tại chương trình con dưới dạng hàm**, **không có thủ tục.**
  - Giá trị hàm có thể không cần dùng đến
  - Có thể không có giá trị nào gán vào tên hàm (void)
  - Cung cấp các giá trị không phải là vô hướng

# 4.2 Cách xây dựng hàm

## Cú pháp

```
<kiểu trả về> <tên hàm>([danh sách tham số])
{
        <các câu lệnh>
        [return <giá trị>;]
}
```



# Một số quy tắc

- Tham số thực sự và tham số hình thức
  - Tham số hình thức: tham số dùng khi khai báo
  - Tham số thực sự: tham số được cung cấp cho hàm khi được sử dụng
  - Tham số thực sự có thể là một biểu thức còn tham số hình thức thì không thể là một biểu thức
- Lệnh return
  - Tương đương lệnh `<Tên hàm> = <Giá trị>`
  - return có thể trả lại giá trị cả một biểu thức

    **Ví dụ:** `return x*x + b*x + c`
  - return có thể xuất hiện nhiều lần trong hàm

```
Ví dụ: if (s>0) return (s);
       else return (-s);
```

# Một số quy tắc (tt)

- Hàm không trả lại giá trị
  - Dùng từ khoá **`void`** để khai báo (Ví dụ 1)
- Hàm không có tham số
  - Khai báo: `Tên_hàm(void)`
  - Ví dụ: Nhập số nguyên, trả về giá trị số nhập vào

```
int Nhap()
{
        int n;
        printf("Nhap mot so nguyen: ");
        scanf("%d", &n);
        return n;
}
```

# Một số quy tắc (tt)

- Hàm phải được khai báo và định nghĩa trước khi sử dụng và thường đặt ở trên hàm chính (hàm main).
- Ví dụ:

```
int Tong(int a, int b)
{
return a + b;
}
void main()
{
int a = 2912, b = 1706;
int sum = Tong(a, b);    /* Loi goi ham */
```

# Một số quy tắc (tt)

- Thông thường, trước hàm main ta chỉ xác định tên hàm, các tham số và giá trị trả về, phần định nghĩa sẽ được đưa xuống dưới cùng. Phần này được gọi là nguyên mẫu hàm (function prototype).

```
int Tong(int a, int b); // prototype ham Tong
void main()
{
int a = 2912, b = 1706;
int sum = Tong(a, b);    /* Loi goi ham */
}
int Tong(int a, int b)   /* Mo ta ham tong */
{
return a + b; }
```

# Một số ví dụ

- Ví dụ 1: Chuyển chữ thường thành chữ hoa

```c
#include <stdio.h>
#include <conio.h>
char to_UpperCase(char ch)
{
    if (ch>='a' && ch <='z')
        return (ch + 'A'-'a');
    else return ch;
}

int main()
{   char thuong,hoa;
    printf("Nhap vao mot ki tu: ");
    scanf("%c", &thuong);
    hoa = to_UpperCase(thuong);
    printf("\nChu hoa tuong ung la: %c\n",hoa);
    getch();
}
```

# 4.3 Tầm tác dụng của biến

- Các loại biến
  - Toàn cục: khai báo trong ngoài tất cả các hàm và có tác dụng lên toàn bộ chương trình.
  - Cục bộ: khai báo trong hàm hoặc khối { } và chỉ có tác dụng trong bản thân hàm hoặc khối đó (kể cả khối con nó). Biến cục bộ sẽ bị xóa khỏi bộ nhớ khi kết thúc khối khai báo nó.

```
#include <stdio.h>
int i; /*Bien toan cuc */
main()
{ … }
void thi_du()
{
        int m=3; /*Bien cuc bo */
}
```

# Chú ý

- Cấp phát bộ nhớ tĩnh cho biến cục bộ:

```
static <tên kiểu> <tên biến>;
```

- Khai báo kiểu bố trí ô nhớ cho biến int nào đó được sử dụng rất nhiều là kiểu bộ nhớ thanh ghi **register** để tăng tốc độ xử lý. Biến thanh ghi thường là các biến đếm trong một vòng lặp nào đó.
  - ***Ví dụ:***

```
register int t;
for (t=0; t<1000; t++)
printf("Lan goi thu %d",t);
```

# 4.4 Truyền tham số cho hàm

- Trong C thực hiện truyền tham số theo một kiểu duy nhất: truyền giá trị

```
#include <stdio.h>
void hoan_vi(int a, int b); /* prototype */
main()
{       int n = 10, p=20;
        printf("Truoc khi goi ham: %d  %d\n",n,p);
        hoan_vi(n,p);
        printf("Sau khi goi ham: %d  %d\n",n,p);
}
void hoan_vi(int a, int b)
{       int t;
        printf("Truoc khi hoan vi: %d  %d\n",a,b);
        t=a; a=b; b=t;
        printf("Sau khi hoan vi: %d  %d\n",a,b);
}
```

# Truyền tham số cho hàm

- Truyền địa chỉ cho hàm
  - ***a** là giá trị được lưu trữ trong bộ nhớ có địa chỉ **a**
  - &a là địa chỉ bộ nhớ chứa giá trị **a**

```
void hoan_vi(int *a, int *b);
main()
{       int n=10, p=20;
        printf("Truoc khi goi ham: %d  %d\n",n,p);
        hoan_vi(&n,&p);
        printf("Sau khi goi ham: %d %d",n,p);
}
void hoan_vi(int *a, int *b)
{       int t;
        printf("Truoc khi hoan vi: %d  %d\n",*a,*b);
        t=*a;    *a=*b;    *b=t;
        printf("Sau khi hoan vi: %d  %d\n",*a,*b);
}
```

# Lưu ý khi truyền đối số

- Lưu ý
  - Trong một hàm, các tham số có thể truyền theo nhiều cách.

```
void HonHop(int x, int *y)
{
        ...
}
```

  - **KHÔNG** được truyền giá trị cho tham số *a và *b trong ví dụ trên, ví dụ, không viết: `hoan_vi(1,5)`
  - Truyền giá trị được sử dụng khi không có nhu cầu thay đổi giá trị của tham số sau khi thực hiện hàm
  - Truyền địa chỉ hoặc truyền tham biến được sử dụng khi có nhu cầu thay đổi giá trị của tham số sau khi thực hiện hàm

# Lời gọi hàm

- Cách thực hiện
  - Gọi tên của hàm đồng thời truyền các đối số (hằng, biến, biểu thức) cho các tham số theo đúng thứ tự đã được khai báo trong hàm.
  - Các biến hoặc trị này cách nhau bằng dấu ,
  - Các đối số này được được đặt trong cặp dấu ngoặc đơn ( )

```
<tên hàm> (<đối số 1>,… , <đối số n>);
```

# 4.6 Một số hàm thông dụng

- Các hàm toán học (trong **stdlib.h**)
  - **int abs(int x);** giá trị tuyệt đối của số nguyên x
  - **long int labs(long int x);** giá trị tuyệt đối của số nguyên dài x
  - **double fabs(double x);**
  - **int rand(void);** cho giá trị ngẫu nhiên từ 0 đến 32767
  - **int random(int n);** cho một giá trị ngẫu nhiên từ 0 đến n-1
  - **void srand(unsigned seed);** khởi đầu bộ số ngẫu nhiên bằng giá trị seed
  - **void randomize(void);** tạo điểm xuất phát ngẫu nhiên cho các hàm rand() và random() ở trên

# Một số hàm thông dụng (tt)

- Các hàm toán học
  - ***Các hàm lượng giác***: `sin, asin, sinh, cos, cosh, acos, tan, atan, tanh`
  - `double log(double x);` tính logarit tự nhiên của x
  - `double log10(double x);` tính logarit cơ số 10 của x
  - `double pow(double x, double y);` tính $x^y$
  - `double ceil(double x);` hàm làm tròn lên, trả về số nguyên nhỏ nhất lớn hơn x

  Ví dụ: `ceil(123.54) = 124`
  - `double floor(double x);` hàm làm tròn xuống, cắt phần lẻ.

  Ví dụ: `floor(123.54) = 123`

# Một số hàm thông dụng (tt)

- Các hàm thời gian
  - **`void getime(struct time *t);`** Trả về giờ hệ thống và đặt vào các thành phần của một biến cấu trúc kiểu time do con trỏ **`t`** trỏ tới.
  - Kiểu cấu trúc **`time`** trong dos.h được định nghĩa:

```
struct time
{  unsigned ti_hour;  /*giờ */
   unsigned ti_min; /*phút*/
   unsigned ti_sec; /*giây */
   unsigned ti_hund; /*phần trăm*/
}
```

  - **`void settime(struct time *t);`** đặt lại giờ hệ thống theo giá trị các thành phần của một cấu trúc kiểu time do con trỏ **`t`** trỏ tới.

# Một số hàm thông dụng (tt)

- Các hàm ngày tháng
  - **`getdate(struct date *d);`** Hàm này nhận ngày hệ thống và đặt vào các thành phần của một biến cấu trúc kiểu **`date`** do con trỏ **`d`** trỏ tới.
  - Kiểu cấu trúc **`date`** được định nghĩa trong dos.h

```
struct date
{       int da_year;
        char da_mon;
        char da_day;
}
```

  - **`void setdate(struct date *d);`** Đặt lại ngày hệ thống theo giá trị các thành phần của một biến cấu trúc kiểu date do con trỏ d trỏ tới.

# Một số hàm thông dụng (tt)

## Hàm chuyển đổi xâu kí tự

- `char *itoa (int x, char *s, int cs);` chuyển đổi số nguyên `x` trong hệ đếm cơ số `cs` sang chuỗi và lưu vào vùng nhớ `s`, trả về địa chỉ vùng `s`.
- `char *ltoa (long x, char *s, int cs);` chuyển đổi số nguyên dài kiểu `long x` trong hệ đếm cơ số `cs` sang chuỗi và lưu vào vùng nhớ `s`, trả về địa chỉ vùng `s`.
- `char *ultoa (unsigned long x, char *s, int cs);` chuyển số kiểu `unsigned long x` trong hệ đếm cơ số `cs` sang chuỗi và lưu vào vùng nhớ `s`, trả về địa chỉ của vùng `s`.
- `double atof (const char *s);` chuyển xâu `str` thành số `float`
- `int atoi (const char*s);` chuyển xâu `str` thành số `int`
- `long atol (cont char *s);` chuyển xâu `str` thành số `long`

# Một số hàm thông dụng (tt)

## Các hàm cấp phát động

- `unsigned coreleft (void);` cho biết số bộ nhớ khả dụng trong vùng cấp phát động đối với mô hình **tiny**, **small** và **medium**
- `unsigned long coreleft (void);` cho biết số bộ nhớ khả dụng trong vùng cấp phát động đối với mô hình **compact large** và **huge**
- `void *calloc (size_t n, size_t size);` cấp phát vùng nhớ cho n đối tượng cỡ **size byte**
- `void *malloc (size_t size);` cấp phát vùng nhớ cho **size byte**
- `void *realloc (void *block, size_t size);` cấp phát lại bộ nhớ
- `void free (void *block);` giải phóng vùng nhớ đã cấp