Bài 9

Lập trình C
Hàm

Nội dung

- Hàm
- Gọi hàm bằng giá trị
- Gọi hàm bằng tham chiếu
- Gọi hàm với các mảng
- Cơ chế kết thúc của hàm
- Nguyên mẫu của hàm
- Hàm đệ qui

Hàm

- Trong ngôn ngữ C, hàm là một khối xây dựng cơ bản của một chương trình C cung cấp tính mô-đun.
- Khối lệnh **định nghĩa của một hàm**

  <kiểu trả về> <tên hàm>(<danh sách tham số>)

  {

  <các lệnh phần thân hàm>

  }
- ***<kiểu trả về>*** xác định kiểu dữ liệu mà hàm trả về. Một hàm có thể trả về bất kỳ kiểu dữ liệu nào, ngoại trừ kiểu mảng.
- ***<danh sách tham số>*** là một danh sách các tên biến và kiểu tương ứng được phân cách bằng dấu phẩy. Dùng để nhận giá trị của các đối số khi hàm được gọi để thực thi.

Hàm - Phạm vi

- Định nghĩa của hàm là một khối mã riêng biệt, do đó nó xác định một phạm vi khối.
  - Các lệnh phần thân hàm là riêng tư đối với một hàm và chúng chỉ được truy cập (thi hành) thông qua lời gọi hàm đó.
  - Biến được khai báo trong một hàm là biến cục bộ. Nó tồn tại khi hàm đang thi hành và bị hủy khi hàm kết thúc.
  - Các tham số của một hàm cũng nằm trong phạm vi của hàm đó. Nó cũng được dùng như biến cục bộ của hàm.
- Tất cả các hàm có phạm vi tập tin. Vì vậy không thể định nghĩa một hàm trong một hàm.

Hàm - Đối số của hàm

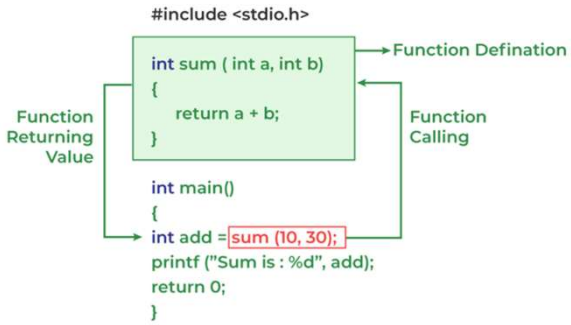
- Khi một hàm thực thi và cần được truyền một mục dữ liệu cho nó thì mục dữ liệu đó được gọi là **đối số của hàm**.
- Nếu một hàm cần đối số để thi hành, phải khai báo **tham số để nhận giá trị của đối số** khi hàm đó được gọi.

```
/* Return 1 if c is part of string s; 0 otherwise. */
int findChar(char *s, char c)
{
    while (*s)
        if (*s == c) return 1;
        else s++;

    return 0;
}
```


Hàm - Lời gọi hàm

- **Lời gọi hàm** (function call) là một biểu thức có dạng
  <tên hàm>(<danh sách đối số>)
  chỉ thị cho trình biên dịch thực thi hàm. Nó chuyển điều khiển chương trình đến định nghĩa hàm.
- Biểu thức gọi hàm có giá trị với kiểu trả về của hàm, hoặc không có giá trị nếu kiểu trả về của hàm là void.



Gọi hàm bằng giá trị

- Truyền đối số bằng giá trị - **Gọi hàm bằng giá trị**:
  - Giá trị của một đối số được sao chép vào tham số của hàm.
  - Bên trong hàm, những thay đổi được thực hiện đối với tham số không ảnh hưởng đến đối số.

```c
/* Return the area of a number. */
int sqrt(int x) {
    x = x * x;
    return x;
}

int main() {
    int x = 10;
    printf("sqrt(%d) = %d;\tx = %d\n", x, sqrt(x), x);

    return 0;
}
```


Gọi hàm bằng tham chiếu (1)

- Truyền đối số bằng tham chiếu - **Gọi hàm bằng tham chiếu**:
  - Địa chỉ của đối số được sao chép vào tham số của hàm.
  - Bên trong hàm, địa chỉ được dùng để thao tác với đối số thực sự được sử dụng trong lời gọi hàm. Nghĩa là những thay đổi được thực hiện với tham số sẽ ảnh hưởng đến đối số.
- Tạo lời gọi hàm bằng tham chiếu
  - Sử dụng **địa chỉ của đối số để truyền cho hàm** thay vì dùng chính đối số đó.
  - Khi đó phải **khai báo tham số dưới dạng kiểu con trỏ**.

Gọi hàm bằng tham chiếu (2)

```c
/* Swap two numbers. */
void swap(int *a, int *b) {
    int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
}

int main() {
    int a = 10, b = 20;
    printf("Before swapping: a = %d;\tb = %d\n", a, b);
    /* Pass the addresses of a and b to the function swap() */
    swap(&a, &b);
    printf("After swapping: a = %d; \tb = %d\n", a, b);
    return 0;
}
```

```
Before swapping: a = 10;        b = 20
After swapping: a = 20;         b = 10
```


Gọi hàm với các mảng (1)

- Khi một mảng được sử dụng làm đối số cho lời gọi hàm thì địa chỉ của nó sẽ được truyền cho hàm. Nghĩa là một con trỏ chỉ mục được truyền cho hàm.
- Mã bên trong hàm có khả năng thay đổi nội dung thực sự của mảng được sử dụng cho lời gọi hàm.

```c
/* Reverse an integer array. */
void reverseArray(int *pArr, int n) {
    for (int i = 0; i < n / 2; i++)
        /* Pass the addresses of two elements i-th and (n - i + 1)-th
           to the function swap() */
        swap(pArr + i, pArr + (n - i - 1));
}
```

Gọi hàm với các mảng (2)

```
int main() {
    int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

    /* Pass the addresses and size of the array a
       to the function reverseArray() */
    reverseArray(a, 10);
    printf("After reversing:\n");
    for (int i = 0; i < 10; i++)
        printf("%5d", a[i]);
    printf("\n");

    return 0;
}
```

```
After reversing:
    9    8    7    6    5    4    3    2    1    0
Press any key to continue . . .
```


Gọi hàm với các mảng (3)

- Có thể dùng kiểu của con trỏ chỉ mục để khai báo kiểu tham số của hàm để nhận đối số là một mảng. Thường sử dụng:
  - Con trỏ trỏ tới phần tử nguyên tử đầu tiên của mảng.
  - Con trỏ trỏ tới phần tử thứ 0 của mảng.
- Vì mảng được truyền bằng tham chiếu (địa chỉ của mảng) nên nó bị mất thông tin về kích thước. Do đó cần thêm tham số để truyền kích thước của mảng cho hàm.
- Khai báo kiểu tham số cho đối số là mảng một chiều bằng
  - Con trỏ trỏ tới phần tử nguyên tử đầu tiên - phần tử thứ 0

  <kiểu trả về> <tên hàm>(<kiểu dữ liệu> *<tên tham số>, …) hoặc

  <kiểu trả về> <tên hàm>(<kiểu dữ liệu> <tên tham số>[ ], …)

```
/* the function sumArray1 can receive
   the n-elements array as an argument. */
int sumArray1(int *pArr, int n)
// or
int sumArray2(int arr[], int n)
```

Gọi hàm với các mảng (4)

- Khai báo kiểu tham số cho đối số là mảng hai chiều bằng
  - Con trỏ trỏ tới phần tử nguyên tử đầu tiên - phần tử (0, 0)

    <kiểu trả về> <tên hàm>(<kiểu dữ liệu> *<tên tham số>, …)

```
/* the function sum2DArray1 can receive
   the (MxN) array as an argument. */
int sum2DArray1(int *pArr, int m, int n)
```

  - Con trỏ trỏ tới phần tử thứ 0: ***<kiểu dữ liệu> (*)[ ]***

    <kiểu trả về> <tên hàm>(
    <kiểu dữ liệu> <tên tham số>[ ][kích thước cột], …)

```
/* the function sum2DArray2 can receive
   the (Mx4) array as an argument. */
int sum2DArray2(int arr[][4], int m, int n)
```


Gọi hàm với các mảng (5)

- Khai báo kiểu tham số cho đối số là mảng ba chiều bằng
  - Con trỏ trỏ tới phần tử nguyên tử đầu tiên - phần tử (0, 0, 0)

    <kiểu trả về> <tên hàm>(<kiểu dữ liệu> *<tên tham số>, …)

```
/* the function sum3DArray1 can receive
   the (LxMxN) array as an argument. */
int sum3DArray1(int *pArr, int l, int m, int n)
```

  - Con trỏ trỏ tới phần tử thứ 0: ***<kiểu dữ liệu> (*)[ ][ ]***

    <kiểu trả về> <tên hàm>(<kiểu dữ liệu>
    <tên tham số>[ ][kích thước hàng][kích thước cột],
)

```
/* the function sum3DArray2 can receive
   the (Lx3x4) array as an argument. */
int sum3DArray2(int arr[][3][4], int l, int m, int n)
```

Gọi hàm với các mảng (6)

```c
/* the function sumArray1 can receive
   the n-elements array as an argument. */
int sumArray1(int *pArr, int n) {
    printf("* The function sumArray1 ...size of the parameter pArr: %zd\n", sizeof(pArr));
    int s = 0;
    for (int i = 0; i < n; i++)
        s += pArr[i];

    return s;
}

/* the function sumArray2 can receive
   the n-elements array as an argument. */
int sumArray2(int arr[], int n) {
    printf("* The function sumArray2 ...size of the parameter arr: %zd\n", sizeof(arr));
    int s = 0;
    for (int i = 0; i < n; i++)
        s += arr[i];

    return s;
}
```


Gọi hàm với các mảng (7)

```c
int main() {
    int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    printf("* Size of the array a: %zd\n", sizeof(a));
    printf("Sum of the elements of the array: %d\n", sumArray1(a, 10));
    printf("Sum of the elements of the array: %d\n", sumArray2(a, 10));

    return 0;
}
```

```
* Size of the array a: 40
* The function sumArray1 ...size of the parameter pArr: 8
Sum of the elements of the array: 55
* The function sumArray2 ...size of the parameter arr: 8
Sum of the elements of the array: 55
```

Gọi hàm với các mảng (8)

```c
/* the function sum2DArray1 can receive
   the (MxN) array as an argument. */
int sum2DArray1(int *pArr, int m, int n) {
    printf("* The function sum2DArray1 ...size of the parameter pArr: %zd\n", sizeof(pArr));
    int s = 0;
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            s += pArr[i * n + j];

    return s;
}

/* the function sum2DArray2 can receive
   the (Mx4) array as an argument. */
int sum2DArray2(int arr[][4], int m, int n) {
    printf("* The function sum2DArray2 ...size of the parameter arr: %zd\n", sizeof(arr));
    int s = 0;
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            s += arr[i][j];

    return s;
}
```


Gọi hàm với các mảng (9)

```c
int main() {
    int b[3][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
    printf("* Size of the 2D array b: %zd\n", sizeof(b));
    printf("Sum of the elements of the 2D array: %d\n", sum2DArray1((int *)b, 3, 4));
    printf("Sum of the elements of the 2D array: %d\n", sum2DArray2(b, 3, 4));

    return 0;
}
```

```
* Size of the 2D array b: 48
* The function sum2DArray1 ...size of the parameter pArr: 8
Sum of the elements of the 2D array: 78
* The function sum2DArray2 ...size of the parameter arr: 8
Sum of the elements of the 2D array: 78
```

Cơ chế kết thúc của hàm (1)

- **Lệnh return** được sử dụng để trả về từ một hàm. Nó kết thúc việc thực thi của hàm và chuyển điều khiển về điểm gọi hàm.

  return <biểu thức>;

- Sử dụng lệnh return
  - Lệnh return không giá trị. Khi đó ***<biểu thức>*** là một biểu thức rỗng. Chỉ dụng trong **hàm có kiểu trả về là void**.
  - Lệnh return có giá trị. Chỉ sử dụng trong **hàm có kiểu trả về khác void**. Khi đó giá trị của ***<biểu thức>*** sẽ là giá trị trả về của hàm.
- Trong một hàm có thể có nhiều lệnh return. Tuy nhiên hàm sẽ ngừng thực thi khi gặp lệnh return đầu tiên.

Cơ chế kết thúc của hàm (2)

- Một hàm kết thúc việc thực thi và trả về điểm gọi hàm theo 2 cách:
  - Khi câu lệnh cuối cùng trong mã của hàm được thực thi.
  - Khi gặp lệnh return đầu tiên.

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

```
int findChar(char *s, char c) {
    while (*s)
        if (*s == c) return 1;
        else s++;

    return 0;
}
```

Cơ chế kết thúc của hàm - Trả về giá trị

- Ngoại trừ các hàm có kiểu trả về là void, tất cả các hàm đều trả về một giá trị. Giá trị này được xác định bởi biểu thức của lệnh return.
- Lời gọi hàm trả về giá trị có thể sử dụng làm toán hạng trong biểu thức.

```
/* Calculate the sum of an integer array. */
int sumArray(int *pArr, int n) {
    int s = 0;
    for (int i = 0; i < n; i++) {
        s += pArr[i];
    }

    return s;
}
```

```
/* the function call sumArray() is operand. */
s = sumArray(a, 10) + sumArray(d, 5);
```


Cơ chế kết thúc của hàm - Trả về con trỏ

- Để trả về một con trỏ, một hàm phải được khai báo với kiểu trả về là một kiểu con trỏ.

```
/* Return a pointer of the first occurrence of the character
   c in the string s. Otherwise, a pointer to the null
   terminator is returned. */
char *searchChar(char *s, char c) {
    while (*s && (c != *s)) s++;
    return s;
}
```

```
char s[] = "Hello", ch = 'l', *p = searchChar(s, ch);
if (*p)
    printf("The first occurrence of %c is %s\n", ch, p);
else
    printf("No match found.\n");
```

Hàm đệ qui (1)

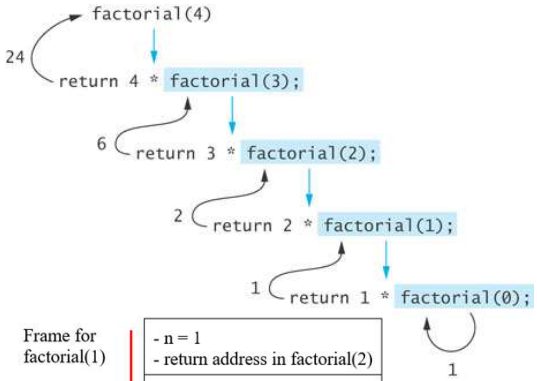
- Trong ngôn ngữ C, một hàm có thể gọi chính nó. Hàm như vậy được gọi là **hàm đệ qui**.
- Khi một hàm gọi chính nó, một tập hợp các biến cục bộ và tham số hình thức mới được cấp phát bộ nhớ trên ngăn xếp và mã hàm được thực thi từ đầu với các biến mới này.

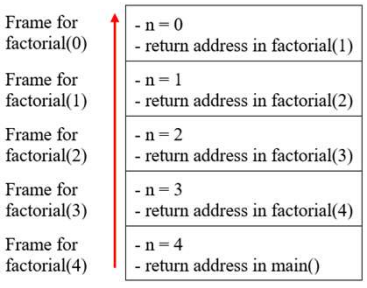
```
/* The recursive function to define the factorial of
   an non-negative integer. */
int factorial(int n) {
    if (n == 0)
        return 1;
    else
        return n * factorial(n - 1);
}
```


Hàm đệ qui (2)

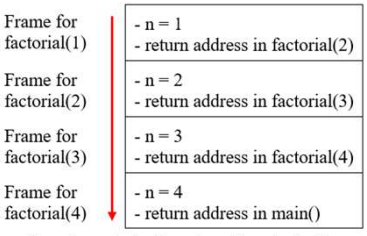
```
int main() {
    int res = factorial(4);

    return 0;
}
```





Run-time stack after all calls



Run-time stack after return from last call