CẤU TRÚC DỮ LIỆU & GIẢI THUẬT 1

ThS. Nguyễn Chí Hiếu

2017

NỘI DUNG

Lập trình C, C++
- Biến con trỏ
- Kiểu dữ liệu cấu trúc (struct)
- Truyền tham số

Đệ quy
- Khái niệm
- Phân loại đệ quy

Biến

Một biến khi được khai báo gồm 3 thuộc tính cơ bản

- Tên biến.
- Giá trị của biến.
- Địa chỉ của biến trong vùng nhớ.

Biến

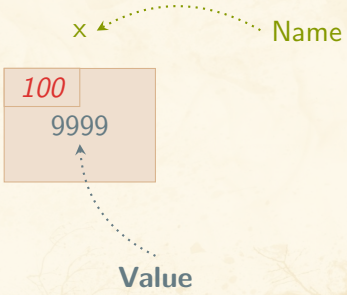
Một biến khi được khai báo gồm 3 thuộc tính cơ bản

- Tên biến.
- Giá trị của biến.
- Địa chỉ của biến trong vùng nhớ.

Biến

Một biến khi được khai báo gồm 3 thuộc tính cơ bản

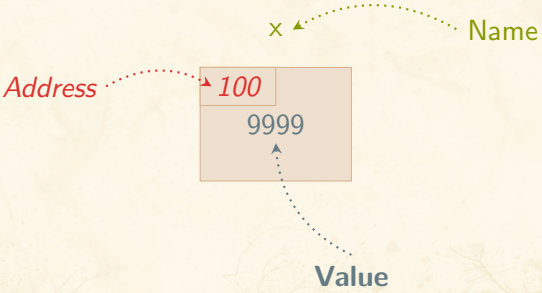
- Tên biến.
- Giá trị của biến.
- Địa chỉ của biến trong vùng nhớ.



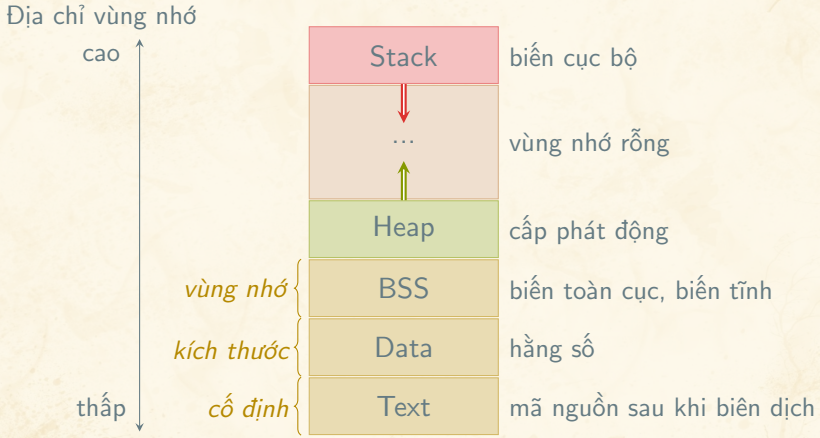
Biến

Một biến khi được khai báo gồm 3 thuộc tính cơ bản

- Tên biến.
- Giá trị của biến.
- Địa chỉ của biến trong vùng nhớ.



Quản lý vùng nhớ trong chương trình



Biến

Cấp phát vùng nhớ của một biến

- Cấp phát tĩnh
  - Biến toàn cục *(global)* và biến tĩnh *(static)*: chạy chương trình.
  - Biến cục bộ *(local)*: gọi hàm.
- Cấp phát động
  - Sử dụng từ khóa `new` hay `malloc()` để tạo một vùng nhớ dữ liệu trong heap.
  - Sử dụng từ khóa `delete` hay `free()` để thu hồi vùng nhớ được cấp phát động.
  - Vùng nhớ cấp phát động trong heap được quản lý thông qua biến con trỏ.

Biến con trỏ

Khái niệm

- Biến con trỏ *(pointer)* là biến lưu địa chỉ của một kiểu dữ liệu nào đó.
- Cú pháp

<Kiểu dữ liệu> *<Tên con trỏ>;

```
1  int      *p1, *p2;
2  float    **p3;
3  Student  *s;
```

Biến con trỏ

Khái niệm

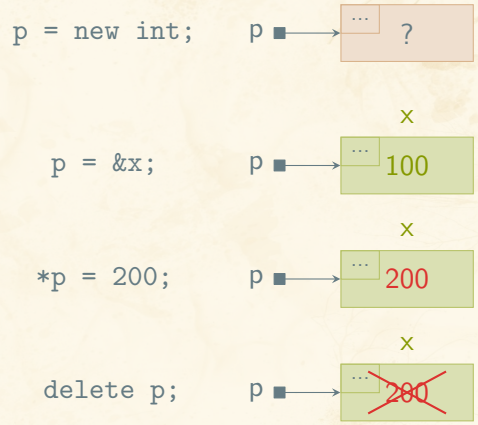
- Con trỏ null là con trỏ chưa trỏ vào đâu cả *(mặc định địa chỉ là 0x00000000)*, khác với con trỏ mới được khai báo và chưa được khởi tạo. Trong C, C++ con trỏ null được gán bằng `NULL` hay `nullptr`.
- Sử dụng từ khóa `new` để cấp phát và và `delete` để thu hồi vùng nhớ biến con trỏ.

Sử dụng biến con trỏ

- Toán tử địa chỉ `&` *(address operator)*: lấy địa chỉ của một biến.
- Toán tử gián tiếp `*` *(indirection operator)* lấy nội dung (giá trị) tại địa chỉ mà con trỏ trỏ đến.

Biến con trỏ

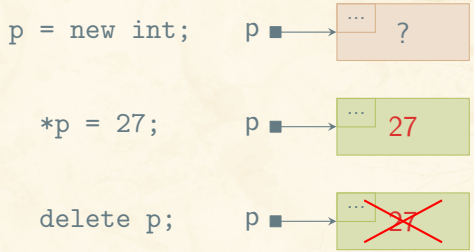
```
1   int x = 100;
2   int *p = new int;
3
4   p = &x;
5
6   cout << x; // 100
7
8   *p = 200;
9
10  cout << x; // 200
11
12  delete p; // error
```



✍ *Con trỏ p đang trỏ đến biến x, không thể dùng từ khóa `delete` để hủy p.*

Biến con trỏ

```
1  int *p = new int;
2
3  *p = 27;
4
5  delete p;
6
7  *p = 2; // error
```



✍ *Con trỏ p đã bị hủy nên không thể gán giá trị tại địa chỉ con trỏ p trỏ đến.*

Biến con trỏ

Sử dụng const và biến con trỏ

- Sử dụng từ khóa `const` nhằm ngăn sự thay đổi giá trị biến con trỏ ngoài ý muốn.
- Khai báo `const` phía trước (*con trỏ trỏ đến hằng*)
  ```
  const int* p1 = &x; // *p1 không được thay đổi
  ```
- Khai báo `const` phía sau (*con trỏ hằng*)
  ```
  int* const p2 = &x; // p2 không được thay đổi
  ```

Biến con trỏ

Sử dụng const và biến con trỏ

```
1  int x = 100;
2  int *p1 = &x;
3  *p1 = 200;
4  p1++;
5  cout << *p1; // ???
6  const int *p2 = &x;
7  int *const p3 = &x;
8  *p2 = 400;  // error
9  p3++;       // error
```

Kiểu dữ liệu cấu trúc (struct)

Khái niệm

- Là một kiểu dữ liệu được kết hợp từ nhiều kiểu dữ liệu khác.
- Cú pháp

```
struct <Tên cấu trúc>
{
        <Kiểu dữ liệu 1> <Thành phần 1>;
        <Kiểu dữ liệu 2> <Thành phần 2>;
        // ...
};
```

Truy xuất các thành phần của cấu trúc

- Biến tĩnh: sử dụng dấu chấm "$\cdot$"
- Biến động: sử dụng dấu mũi tên "$\rightarrow$"

Kiểu dữ liệu cấu trúc (struct)

```
1   struct Date
2   {
3     unsigned day;
4     unsigned month;
5     unsigned year;
6   };
7   Date day1;
8   day1.day = 1;
9   day1.moth = 5;
10  day1.year = 2015;
11  Date *day2 = new Date;
12  day2->day = day1.day;
13  day2->month = day1.month;
14  day2->year = day1.year;
```

Truyền tham số

Truyền tham trị

- Giá trị của tham số *không đổi* sau khi thực hiện hàm.

```
1  void Swap1(int a, int b);
2  void main()
3  {
4    int a = 5, b = 10;
5    Swap1(a, b);
6  }
7  void Swap1(int a, int b)
8  {
9    int temp = a;
10   a = b;
11   b = temp;
12 }
```

```
1  a = 5
2  b = 10
```

Truyền tham số

Truyền tham biến

- Giá trị của tham số *thay đổi* sau khi thực hiện hàm.

```
1  void Swap2(int &a, int &b);
2  void main()
3  {
4    ...;
5    Swap2(a, b);
6  }
7  void Swap2(int &a, int &b)
8  {
9    int temp = a;
10   a = b;
11   b = temp;
12 }
```

```
1  a = 10
2  b = 5
```

Truyền tham số

Truyền con trỏ

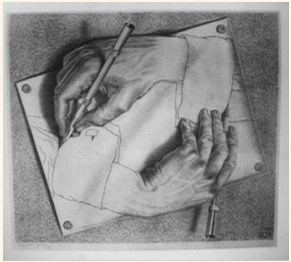
- Giá trị của tham số *thay đổi* sau khi thực hiện hàm.

```
1  void Swap3(int *a, int *b);
2  void main()
3  {
4    ...;
5    Swap3(&a, &b);
6  }
7  void Swap3(int *a, int *b)
8  {
9    int temp = *a;
10   *a = *b;
11   *b = temp;
12 }
```

```
1  a = 10
2  b = 5
```

Đệ quy

Khái niệm



- Vấn đề đệ quy là vấn đề được định nghĩa bằng chính nó.
- Một hàm được gọi là đệ quy, nếu bên trong thân của hàm đó có gọi lại chính nó một cách trực tiếp hay gián tiếp.

Đệ quy

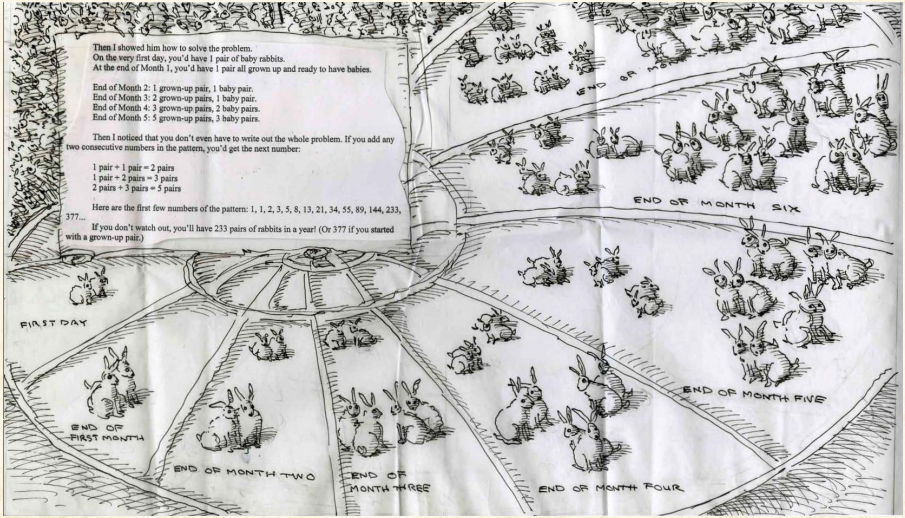
Ví dụ 1

Bài toán nuôi thỏ

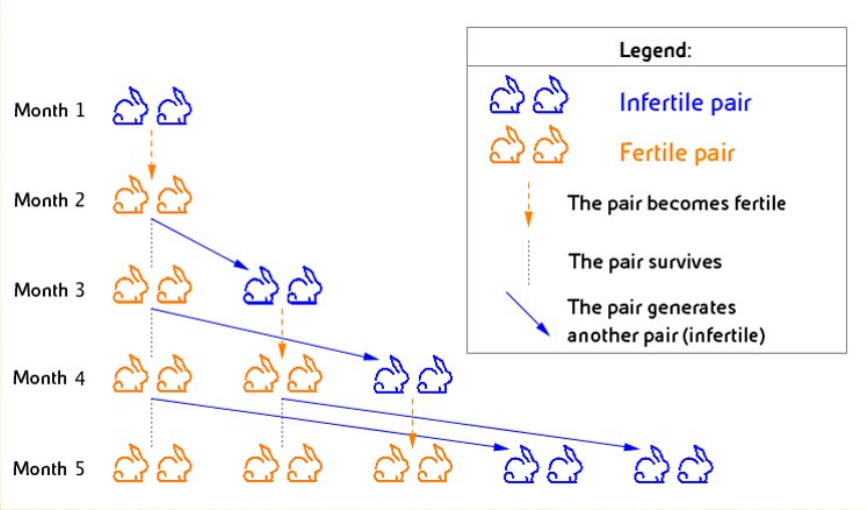
- Bắt đầu với một thỏ đực và một thỏ cái vừa mới chào đời.
- Thỏ đạt tới tuổi sinh sản sau một tháng.
- Thời gian mang thai của một con thỏ là một tháng.
- Sau khi tuổi sinh sản, thỏ cái đẻ đều đều mỗi tháng.
- Một thỏ cái sinh ra một thỏ đực và một thỏ cái.
- Không có thỏ chết.

Hỏi sau một năm sẽ có bao nhiêu cặp thỏ?

Đệ quy


Then I showed him how to solve the problem.
On the very first day, you'd have 1 pair of baby rabbits.
At the end of Month 1, you'd have 1 pair all grown up and ready to have babies.
End of Month 2: 1 grown-up pair, 1 baby pair.
End of Month 3: 2 grown-up pairs, 1 baby pair.
End of Month 4: 3 grown-up pairs, 2 baby pairs.
End of Month 5: 5 grown-up pairs, 3 baby pairs.
Then I noticed that you don't even have to write out the whole problem. If you add any two consecutive numbers in the pattern, you'd get the next number:
1 pair + 1 pair = 2 pairs
1 pair + 2 pairs = 3 pairs
2 pairs + 3 pairs = 5 pairs
Here are the first few numbers of the pattern: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377...
If you don't watch out, you'll have 233 pairs of rabbits in a year! (Or 377 if you started with a grown-up pair.)
FIRST DAY
END OF FIRST MONTH
END OF MONTH TWO
END OF MONTH THREE
END OF MONTH FOUR
END OF MONTH FIVE
END OF MONTH SIX

Đệ quy


Legend:
Infertile pair
Fertile pair
The pair becomes fertile
The pair survives
The pair generates another pair (infertile)
Month 1
Month 2
Month 3
Month 4
Month 5

Đệ quy

Khái niệm

- Hàm đệ quy gồm hai phần
  - *Phần cơ sở*: điều kiện dừng quá trình gọi đệ quy.
  - *Phần đệ quy*: thân hàm chứa lời gọi đệ quy.
- *Bất kỳ một hàm đệ quy nào cũng phải có điều kiện dừng.*

Ví dụ 2

Cho n là số nguyên không âm, tính $n!$ theo công thức

$$f(n) = \begin{cases} 1 & , n = 0 \\ n \cdot f(n-1) & , n > 0 \end{cases}$$

Phân loại đệ quy

Đệ quy tuyến tính

- Trong thân hàm có duy nhất một lời gọi hàm gọi lại chính nó một cách tường minh.

```
1   KieuDuLieu TenHam(ThamSo)
2   {
3     if (DieuKienDung)
4     {
5       ...;
6       return GiaTri;
7     }
8     ...;
9     TenHam(ThamSo);
10  }
```

Phân loại đệ quy

Ví dụ 3

Cho n là số nguyên không âm, $n!$ được định nghĩa như sau

$$f(n) = \begin{cases} 1 & , n = 0 \\ n \cdot f(n-1) & , n > 0 \end{cases}$$

```
1  long Factorial(int n)
2  {
3    if (n == 0)
4      return 1;
5    return n * Factorial(n - 1);
6  }
```

Phân loại đệ quy

Đệ quy nhị phân

- Trong thân hàm có hai lời gọi hàm gọi lại chính nó một cách tường minh.

```
1  KieuDuLieu TenHam(ThamSo)
2  {
3    if (DieuKienDung)
4    {
5      ...;
6      return GiaTri;
7    }
8    ...;
9    TenHam(ThamSo);
10   ...;
11   TenHam(ThamSo);
12 }
```

Phân loại đệ quy

Ví dụ 4

Dãy Fibonacci được định nghĩa như sau

$$f(n) = \begin{cases} 1 & , n = 0, 1 \\ f(n-1) + f(n-2) & , n > 1 \end{cases}$$

```
1  long Fibonacci(int n)
2  {
3    if (n <= 1)
4      return 1;
5    return Fibonacci(n - 1) + Fibonacci(n - 2);
6  }
```

Phân loại đệ quy

Đệ quy phi tuyến

- Trong thân hàm có lời gọi hàm lại chính nó được đặt bên trong thân vòng lặp.

```
1   KieuDuLieu TenHam(ThamSo)
2   {
3     if (DieuKienDung)
4     {
5       ...;
6       return GiaTri;
7     }
8     loop (DieuKieuLap)
9     {
10      ....;
11      TenHam(ThamSo);
12    }
13  }
```

Phân loại đệ quy

Ví dụ 5

Cho hàm $f(n)$ được định nghĩa như sau

$$f(n) = \begin{cases} n & , n \leq 4 \\ f(n-1) + f(n-2) + f(n-3) + f(n-4) & , n > 4 \end{cases}$$

```
1  long F(int n)
2  {
3    int i, result = 0;
4    if (n <= 4)
5      return n;
6    for (i = 1; i <= 4; i ++ )
7      result +=  F(n - i);
8    return result;
9  }
```

Phân loại đệ quy

Đệ quy tương hỗ

- Trong thân hàm 1 có lời gọi hàm tới hàm 2 và bên trong thân hàm 2 có lời gọi hàm đến hàm 1.

```
1   KieuDuLieu TenHam1(ThamSo)
2   {
3     if (DieuKienDung)
4       return GiaTri;
5     TenHam2(ThamSo);
6   }
7   KieuDuLieu TenHam2(ThamSo)
8   {
9     if (DieuKienDung)
10      return GiaTri;
11    TenHam1(ThamSo);
12  }
```

Phân loại đệ quy

Ví dụ 6

Cho số nguyên không âm n. Xác định n là số chẵn hay lẻ.

$$\begin{cases} n & , n = 0, 1 \\ \text{IsOdd}\,(n-1) & , n \neq 0 \\ \text{IsEven}\,(n-1) & , n \neq 1 \end{cases}$$

trong đó, kết quả trả về 1 là số lẻ và 0 là số chẵn.

```
1 bool IsOdd(int n)
2 {
3   if (n == 1)
4     return true;
5   return IsEven(n - 1);
6 }
```

```
1 bool IsEven(int n)
2 {
3   if (n == 0)
4     return false;
5   return IsOdd(n - 1);
6 }
```

Bài tập

Cài đặt hàm đệ quy cho các bài toán

1. Tính tổng của n số nguyên dương đầu tiên

$$S(n) = 1 + 2 + 3 + \cdots + n - 1 + n, n > 0.$$

2. Tính tổ hợp chập k của n phần tử

$$C_n^k = \begin{cases} 1 & , k = 0 \vee k = n \\ C_{n-1}^k + C_{n-1}^{k-1} & , 0 < k < n \end{cases}$$

3. Tính $f(n)$

$$f(n) = \begin{cases} n & , n \leq 3 \\ f(n-1) + 2f(n-2) + 3f(n-3) & , n > 3 \end{cases}$$

Tài liệu tham khảo

Dương Anh Đức, Trần Hạnh Nhi.
Nhập môn Cấu trúc dữ liệu và Thuật toán.
Đại học Khoa học tự nhiên TP Hồ Chí Minh, 2003.

Donald E. Knuth.
The Art of Computer Programming, Volume 3.
Addison-Wesley, 1998.

Niklaus Wirth.
Algorithms + Data Structures = Programs.
Prentice-Hall, 1976.

Robert Sedgewick.
Algorithms in C.
Addison-Wesley, 1990.