Con trỏ hàm và khuôn mẫu

Trần Việt Khoa
tvkhoa.husc@gmail.com,
Khoa Công nghệ thông tin, Đại học Khoa học Huế

Ngày 17 tháng 2 năm 2017

1 Đặt vấn đề

Con trỏ là một kỹ thuật cao cấp trong lập trình, nó được sử dụng để khai thác việc sử dụng bộ nhớ hiệu quả nhất, ngoài ra nó còn có các chức năng như tổ chức dữ liệu, tối ưu hóa việc viết hàm với việc sử dụng linh hoạt tham số của hàm. Bài viết này trình bày một kỹ thuật cơ bản của lập trình con trỏ đó là con trỏ hàm đồng thời dẫn dắt người dùng hướng đến sử dụng một khái niệm trong lập trình hướng đối tượng đó là template.

Bài toán minh họa cho bài viết là cài đặt thuật toán sắp xếp nổi bọt, đây là một thuật toán sắp xếp cơ bản rất dễ hiểu về mặt ý tưởng cũng như cài đặt (C, Pascal). Tuy là một thuật toán kém hiệu quả so với các thuật toán sắp xếp khác nhưng vấn đề đặt ra ở bài thực hành này không phải là thuật toán mà là kỹ thuật lập trình. Ở đây theo trình tự lần lượt ta sẽ cải tiến thuật toán theo từng yêu cầu và qua mỗi lần cải tiến ta sẽ nắm thêm một số kỹ thuật cơ bản về lập trình như:

- Dùng tham số.

- Kiểu dữ liệu của tham số.

- Truyền đối số cho hàm bằng con trỏ hàm.

- Sử dụng khuôn dạng (template) cho hàm.

Và cuối cùng là dùng một hàm đã cài đặt sẵn có đầy đủ các thông số mà ta vừa học.

Thuật toán sắp xếp nổi bọt cho một dãy các phần tử $a_1, a_2, \ldots, a_n$ cho trước có nhiều cách viết, tuy nhiên sau khi tối ưu hóa thuật toán được viết dưới dạng giả mã như sau, tham khảo tại [1]:

```
procedure bubbleSort( A : list of sortable items )
    n = length(A)
    repeat
       newn = 0
       for i = 1 to n-1 inclusive do
          if A[i-1] > A[i] then
             swap(A[i-1], A[i])
             newn = i
          end if
       end for
       n = newn
```

```
    until n = 0
end procedure
```

2 Giải quyết vấn đề

Dựa theo nội dung giả mã của thuật toán ta có thể dễ dàng cài đặt bản đầu tiên (Ver1) cho thuật toán trên như sau, trong trường hợp này dữ liệu giả định là số nguyên kiểu int:

```
void BubbleSort(int arr[], int length){
     int n = length;
     while(n > 0){
      int newn = 0;
      for(int i = 1; i < n; i++)
         if (arr[i-1]> arr[i]){
            std::swap(arr[i-1], arr[i]);
            newn = i;
         }
      n = newn;
   }
}
//Call in main function
int A[] = {1, 2, 5, 7, 6, 1, 3, 2, 9, 15, 14, 20};
int n=12;
BubbleSort(A, n);
```

2.1 Cải tiến lần thứ nhất

Dựa vào bản cài đặt trên, chúng ta đưa ra các yêu cầu cải tiến sau cho hàm:

1. Sắp được với nhiều phần tử hơn bằng cách khai thác vùng nhớ động, để giải quyết ở đây ta dùng con trỏ như tham số cho mảng dữ liệu cần sắp.

2. Một option được đưa vào đó là có thể sắp theo thứ tự tăng (ascending) hoặc giảm (descending), để giải quyết người ta sử dụng con trỏ trỏ đến hàm thứ tự sắp như một tham số trong hàm BubbleSort còn được gọi là **Con trỏ hàm**. Sau đó tùy theo yêu cầu sắp xếp mà truyền vào hàm sắp tương ứng.

Bản cài đặt sau (Ver2) được cải tiến từ bản thứ nhất với hai yêu cầu trên.

```
void BubbleSort(int * arr, int length, int (*pComparison)(int, int)){
     int n = length;
     while(n > 0){
      int newn = 0;
      for(int i = 1; i < n; i++)
         if (pComparison(arr[i-1], arr[i])>0){
            std::swap(arr[i-1], arr[i]);
            newn = i;
         }
      n = newn;
   }
}
//Sap xep tang
int Ascending(int a, int b){
```

```
    return a > b;
}
//Sap xep giam
int Descending(int a, int b) {
    return b > a;
}
//Call in main function
int A[] = {1, 2, 5, 7, 6, 1, 3, 2, 9, 15, 14, 20};
int n=12;
BubbleSort(A, n, Ascending); // neu sap tang
BubbleSort(A, n, Descending); //neu sap giam
```

Nhận xét:

Một tham số được đưa vào cho hàm là: int (*pComparison) (int, int) có kiểu int là địa chỉ của hàm được truyền vào, như vậy khi viết hàm truyền vào (như Accending, Descending) thì tham số hai hàm này phải quan hệ 1-1 với tham số của *pComparison.

Bản cài đặt hàm BubbleSort cải tiến chỉ thay việc so sánh *if (arr[i-1]> arr[i])* bằng *if (pComparison(arr[i-1], arr[i])>0)*.

Bài tập 1: Viết hàm sắp số chẵn trước, số lẻ sau và tăng dần cho dãy số nguyên.

Giải: Ta chỉ viết hàm so sánh theo yêu cầu trên mà vẫn giữ nguyên bản cài đặt BubbleSort như sau:

```
int EvensFirst(int a, int b){
  if ((a % 2) && !(b % 2))
      return 1;
  if (!(a % 2) && (b % 2))
      return 0;
  return Ascending(a, b);
}
//Call in main function
BubbleSort(A, n, EvensFirst);
```

2.2 Cải tiến lần thứ hai

Hàm trên vẫn chưa sắp với mảng có dữ liệu bất kỳ. Vấn đề cải tiến tiếp theo là viết lại hàm trên sao cho có thể sắp cho mảng với dữ liệu bất kỳ.

Vấn đề đặt ra trên được giải quyết bằng cách sử dụng con trỏ không định kiểu: void *. Khi đó ngoài tham số là kích thước của dãy, ta có dùng thêm tham số chỉ kích thước dữ liệu của phần tử trong dãy.

Hàm so sánh cũng phải thay đổi, thay vì được ấn định kiểu như trước, hàm này dùng hai đối là hai con trỏ void. int (*pComparison)(const void*, const void*)) để có thể so sánh cho kiểu dữ liệu bất kỳ. Vì vậy ta phải xác định địa chỉ cụ thể của phần tử dữ liệu cần so sánh, dịa chỉ truy cập đến hai phần tử a[i-1] và a[i] được tính:

```
char *data_i_1 = (char*)arr + type_size * (i-1);
char *data_i  = (char*)arr + type_size * i;
```

Ngoài ra hàm Swap không dùng của std nữa mà phải viết lại như sau:

```
void swap(void *a, void *b, size_t type_size) {
```

```
    void *tmp = malloc(type_size);
    memcpy(tmp, a, type_size);
    memcpy(a,  b, type_size);
    memcpy(b, tmp, type_size);
    free(tmp);
}
```

Phiên bản cụ thể (Ver3) như sau:

```
int Ascending(const void *x, const void *y){
    int a = *(const int *)x;
    int b = *(const int *)y;
    return a - b;
}

int Descending(const void *x, const void *y){
    int a = *(const int *)x;
    int b = *(const int *)y;
    return b - a;
}

void swap(void *a, void *b, size_t type_size) {
    void *tmp = malloc(type_size);
    memcpy(tmp, a, type_size);
    memcpy(a,  b, type_size);
    memcpy(b, tmp, type_size);
    free(tmp);
}
// Khai bao nguyen mau (prototype) nhu ham qsort cua thu vien stdlib.h>. Ban
    co the su dung ham qsort sau khi viet ham nay.
void BubbleSort(void *arr, size_t length, size_t type_size, int
    (*pComparison)(const void*, const void*)) {

     int n = length;
     while(n > 0){
      int newn = 0;
      for(int i = 1; i < n; i++){
            char *data_i_1 = (char*)arr + type_size * (i-1);
            char *data_i = (char*)arr + type_size * i;
            if(pComparison(data_i_1, data_i) > 0){
                swap(data_i_1, data_i, type_size);
            newn = i;
         }
        }
      n = newn;
   }
}
//Call in main function
BubbleSort(A,n , sizeof(*A), Ascending);
```

Nhận xét:

Với cải tiến như trên ta có thể dùng hàm BubbleSort vừa viết tương tự như cách dùng của hàm qsort. Bản cài đặt ở trên cho ta thấy sức mạnh, tính linh hoạt của con trỏ (pointer) một khái niệm không thể thiếu trong kỹ thuật lập trình.

Tuy nhiên theo sự phát triển của ngôn ngữ, kỹ thuật lập trình một khái niệm mới được đưa ra nhằm thay đổi cách sử dụng trên đó là template. Phiên bản cài đặt sau cho thấy

sự thay đổi đó.

```
template<typename T>
void BubbleSort(T *arr, int length, int (*pComparison)(const T& a, const T&
    b))
{
   int n = length;
   while(n > 0)
   {
      int newn = 0;
      for(int i = 1; i < n; i++)
         if (pComparison(arr[i-1], arr[i]) > 0)
         {
            std::swap(arr[i-1], arr[i]);
            newn = i;
         }
      n = newn;
   }
}
int asc(const int &a, const int &b){
     return a > b;
}
int asc(const double &a, const double &b){
     return a > b;
}
int asc(const string& a, const string& b){
     return a.compare(b) > 0;
}

int main()
{
  int iarray[7] = {7, 5, 4, 3, 9, 8, 6};
  double darray[5] = {4.3, 2.5, -0.9, 10.2, 3.0};
  string sarray[14] ={"hom", "nay", "troi", "nang", "chang", "chang", "chu",
      "meo", "di", "hoc", "chang", "mang", "cai", "gi"};

  printf("Sap day so nguyen:\n ");
  BubbleSort(iarray, 7,asc );
  printf("Sap day so thuc:\n ");
  BubbleSort(darray, 5,asc );
  printf("Sap day cac chuoi:\n ");
  BubbleSort(sarray, 14, asc );
  return 0;
}
```

Sau khi vượt qua các yêu cầu để cài đặt cho thuật toán BubbleSort. Để kiểm chứng lại mức độ nắm bắt của mình các bạn hãy thử sức mình khi cài đặt cho thuật toán sau InsertionSort, xem [2] theo từng bước như đã làm ở trên.

3 Ứng dụng

Các cài đặt trên chỉ để tìm hiểu phương pháp cài đặt một hàm với độ linh hoạt cao, giúp người lập trình có thể sử dụng tốt hàm trong nhiều tình huống. Các thư viện chuẩn của các ngôn ngữ lập trình đều cài đặt các hàm này với độ phức tạp tốt nhất. bạn có thể tìm hiểu:

- Hàm qsort của thư viện <stdlib.h>

- Hàm sort của thư viện STL của C++.

Các hàm này được sử dụng rất nhiều trong các bài toán thi lập trình và sẽ được đề cập đến trong một bài viết khác.

Tài liệu

[1] http://en.wikipedia.org/wiki/Bubble_sort

[2] http://en.wikipedia.org/wiki/Insertion_sort