**Cấu trúc dữ liệu + Giải thuật = Chương trình**

Với một cấu trúc dữ liệu đã chọn, sẽ có những giải thuật tương ứng, phù hợp. Khi cấu trúc dữ liệu thay đổi thường giải thuật cũng phải thay đổi theo để tránh việc xử lý gượng ép, thiếu tự nhiên trên một cấu trúc không phù hợp. Hơn nữa, một cấu trúc dữ liệu tốt sẽ giúp giải thuật xử lý trên đó có thể phát huy tác dụng tốt hơn, vừa đáp ứng nhanh vừa tiết kiệm vật tư, giải thuật cũng dễ hiễu và đơn giản hơn.

**Ví dụ 1:** Một chương trình quản lý điểm thi của sinh viên cần lưu trữ các điểm số của 3 sinh viên. Do mỗi sinh viên có 4 điểm số ứng với 4 môn học khác nhau nên dữ liệu có dạng bảng như sau:

Chỉ xét thao tác xử lý là xuất điểm số các môn của từng sinh viên.

Giả sử có các phương án tổ chức lưu trữ sau:

Phương án 1 : Sử dụng mảng một chiều

Có tất cả 3(SV)\*4(Môn) = 12 điểm số cần lưu trữ, do đó khai báo mảng *result* như sau :

int result [ 12 ] = {7, 9, 5, 2, 5, 0, 9, 4, 6, 3, 7, 4};

Khi đó trong mảng *result* các phần tử sẽ được lưu trữ như sau

Và truy xuất điểm số môn j của sinh viên i – là phần tử tại (dòng i, cột j) trong bảng – phải sử dụng một công thức xác định chỉ số tương ứng trong mảng result:

bảngđiểm(dòng i, cột j) Þ result[((i-1)\*số cột) + j]

Ngược lại, với một phần tử bất kỳ trong mảng, muốn biết đó là điểm số của sinh viên nào, môn gì, phải dùng công thức xác định sau

result[ i ] Þ bảngđiểm (dòng((i / số cột) +1), cột (i % số cột) )

Với phương án này, thao tác xử lý được cài đặt như sau :

*void XuatDiem() //Xuất điểm số của tất cả sinh viên  
{*

*const int so\_mon = 4;  
int sv,mon;  
for (int i=0; i<12; i+)  
{*

*sv = i/so\_mon;  
mon = i % so\_mon;  
printf(“Điểm môn %d của sv %d là: %d”, mon, sv,result[i]);*

*}*

*}*