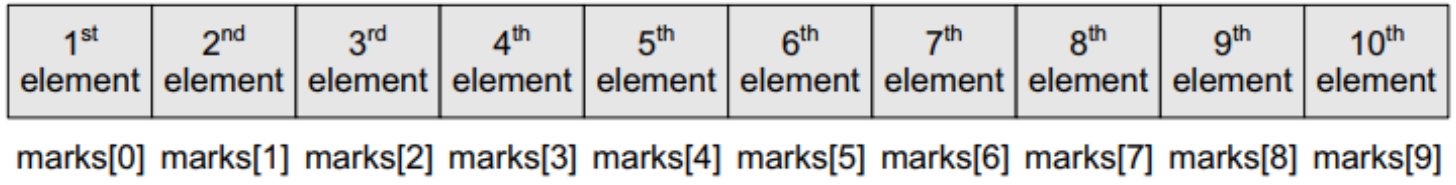
Bài 2. Cấu trúc mảng

Nội dung

- Giới thiệu cấu trúc mảng
- Thao tác trên mảng
- Mảng hai chiều
- Ma trận thưa
- Ứng dụng của mảng
- Bài tập

Giới thiệu cấu trúc mảng

- Mảng là một cấu trúc dữ liệu tuyến tính quan trọng, thường dùng.
- Ví dụ: điểm của các sinh viên, thành tích của các vận động viên, nhiệt độ các ngày trong năm,…
- Đặc trưng:
  - Nhiều ô nhớ kích thước bằng nhau
  - Các ô nhớ liên tục
  - Truy xuất trực tiếp



- Địa chỉ ô nhớ mảng
  - Địa chỉ của ô nhớ A[k] = BA(A) + w(k – lower_bound)
  - Ví dụ: Địa chỉ của marks[4] = 1000 + 2(4 – 0) = 1008.

99	67	78	56	**88**	90	34	85
marks[0]	marks[1]	marks[2]	marks[3]	**marks[4]**	marks[5]	marks[6]	marks[7]
1000	1002	1004	1006	**1008**	1010	1012	1014

Thao tác mảng

- Duyệt mảng: duyệt theo chỉ số.

Thuật toán: duyệt mảng Arr
index = lower_bound
Lặp khi index <= upper_bound
  Thao tác với phần tử Arr[index]
  index = index + 1

- Ví dụ:

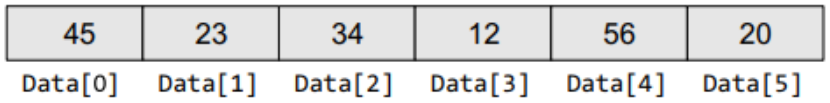
```
for(int i = 0; i < n; i++)
      cout << arr[i] ;
```

Chèn 1 phần tử vào mảng

Input: Mảng A có n phần tử, phần tử cần chèn e, vị trí chèn k
Output: Mảng A sau khi chèn
Action:
Lặp i từ n-1 giảm đến k:
  A[i+1] = A[i]
A[k] = e
n = n + 1

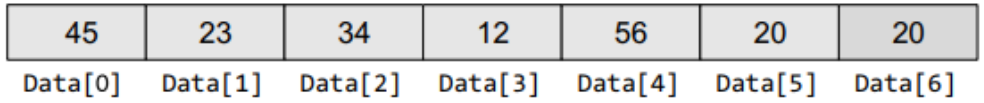
```
void insert(int A[], int &n, int e, int k)
{
   for(int i = n – 1; i >= k; i--)
      A[i+1] = A[i];
   A[k] = e;
   n++;
}
```

Độ phức tạp tính toán của thuật toán là O(n)

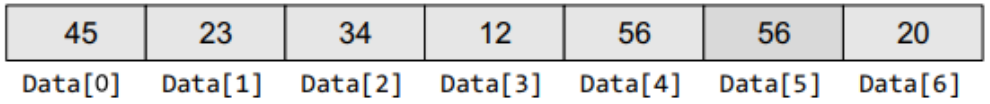


Cần chèn phần tử 100 vào vị trí 3 của mảng Data.

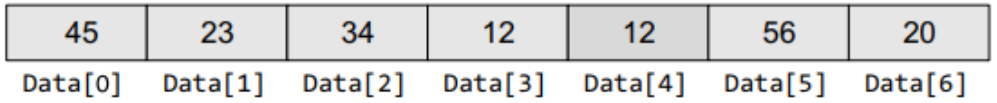
Data[6] = Data[5]



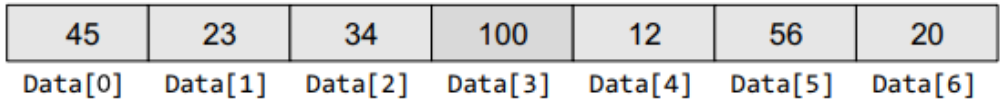
Data[5] = Data[4]



Data[4] = Data[3]



Data[3] = 100

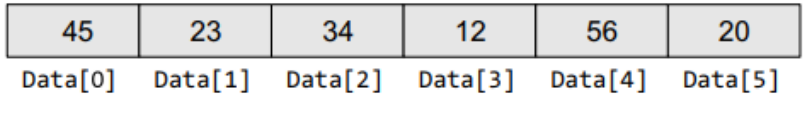


Xóa một phần tử trong mảng

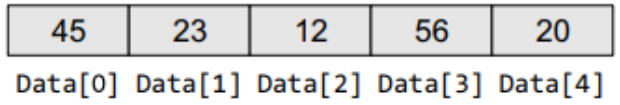
Input: Mảng A có n phần tử, vị trí phần tử cần xóa k
Output: Mảng A sau khi xóa
Action:
Lặp i từ k+1 đến n-1:
    A[i-1] = A[i]
n = n - 1

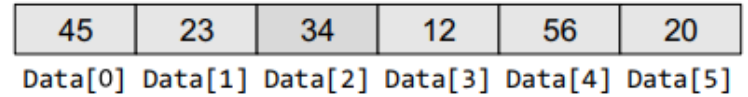
```
void detele(int A[], int &n, int k)
{
   for(int i = k + 1; i < n; i++)
      A[i-1] = A[i];
   n--;
}
```

Độ phức tạp tính toán của thuật toán là O(n)

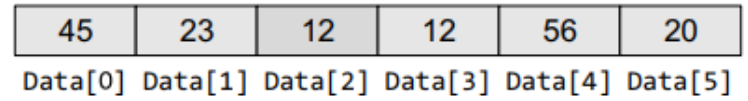


Cần xóa phần tử ở vị trí 3 của mảng Data.

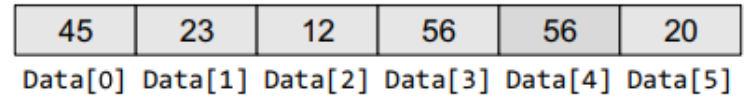




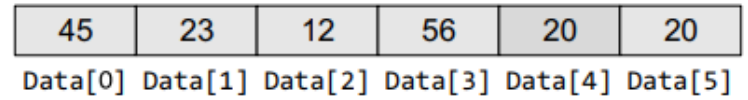
Data[2] = Data[3]



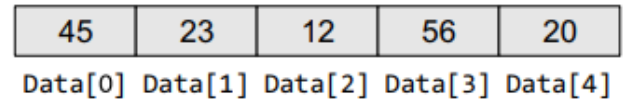
Data[3] = Data[4]



Data[4] = Data[5]



size = size - 1



- Tìm một phần tử trong mảng
  - Thuật toán tìm tuần tự
  - Thuật toán tìm nhị phân

Thuật toán tìm tuần tự

Input: Mảng A có n phần tử, phần tử cần tìm e

Output: True nếu e có trong A, False trong trường hợp ngược lại

Action:

Lặp i từ 0 đến n-1:

nếu A[i] = e thì trả về True

Trả về False

```
bool sequential_search(int A[], int n, int e)
{
  for(int i = 0; i < n; i++)
     if (A[i] == e)
        return true;
  return false;
}
```

Độ phức tạp thuật toán là O(n)

Thuật toán tìm nhị phân

Input: Mảng A có n phần tử đã được sắp tăng, phần tử cần tìm e

Output: True nếu e có trong A, False trong trường hợp ngược lại

Action:

left = 0, right = n - 1

Lặp khi left <= right:

m = (left + right)/2

nếu A[m] = e thì trả về True

ngược lại:

nếu A[m] < e thì:

left = m + 1

ngược lại:

right = m - 1

Trả về False

```
bool binary_search(int A[], int n, int e)
{
   int left = 0, right = n – 1, m;
   while (left <= right)
   {
      m = (left + right)/2;
      if (A[m] == e)
      return true;
      else
      if (A[m] < e)
              left = m + 1;
      else
              right = m – 1;
   }
   return false;
}
```

độ phức tạp thuật toán là O(logn)

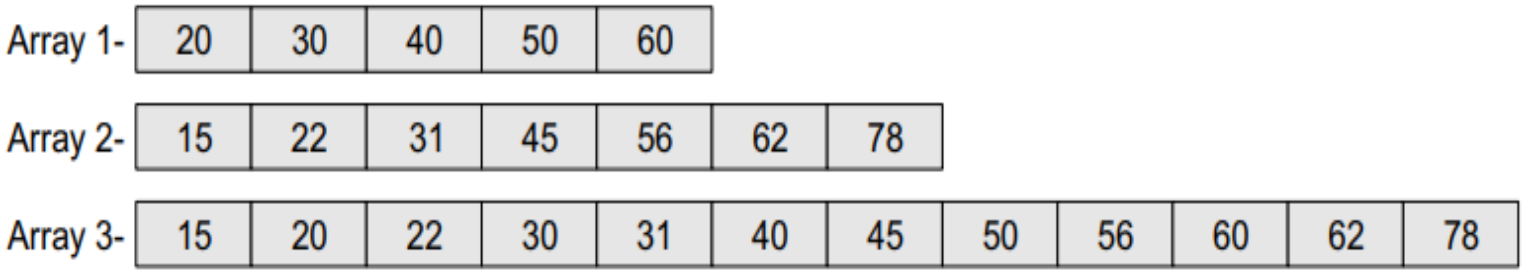
- Sắp xếp mảng

Thuật toán sắp xếp nổi bọt:
Input: mảng A có n phần tử
Output: mảng A sắp tăng
Action:
Lặp i từ 1 đến n – 1:
  Lặp j từ 1 đến n – i:
    Nếu A[j] < A[j-1] thì đổi hai phần tử A[j] và A[j-1]

Độ phức tạp thuật toán sắp xếp nổi bọt là $O(n^2)$

```
void BubbleSort(int A[], int n)
{
 int i, j, tmp;
 for (i = 1; i < n; i++)
    for (j = 1; j <= n-i; j++)
       if (A[j] < A[j-1])
       {
       tmp = A[j];
       A[j] = A[j-1];
       A[j-1] = tmp;
       }
}
```

- Trộn hai mảng



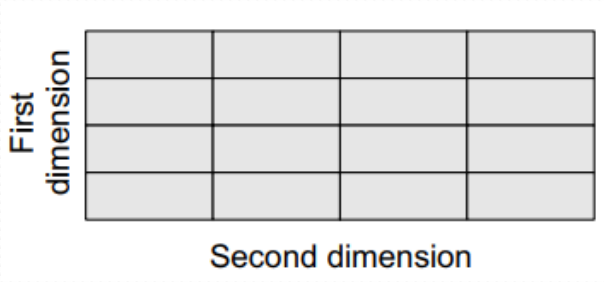
Input: Hai mảng cần trộn A, B đã được sắp
Output: Mảng C trộn của hai mảng A và B và được sắp
Action:
Duyệt lần lượt từng phần tử của hai mảng A và B:
Nếu phần tử ở mảng nào nhỏ hơn thì đưa vào mảng C rồi chuyển sang phần tử tiếp theo
Đưa lần lượt những phần tử của mảng còn lại vào mảng C.

Độ phức tạp của thuật toán trộn hai mảng theo thứ tự là O(m+n)

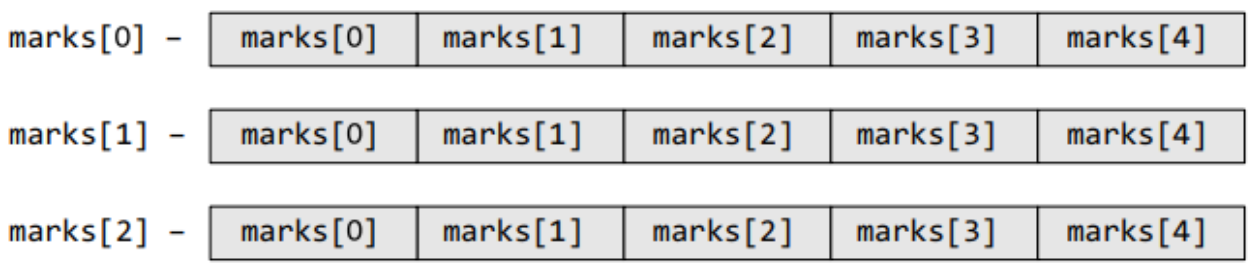
```
void merge_sorted(int A[], int B[], int
C[], int m, int n)
{
  int i = 0, j = 0, k = 0;
  while (i < m && j < n)
  {
     if (A[i] < B[j])
        C[k++] = A[i++];
     else
        C[k++] = B[j++];
  }
  while(i < m) C[k++] = A[i++];
  while(j < n) C[k++] = B[j++];
}
```

Mảng hai chiều

- Mảng mà mỗi phần tử là một mảng.




```
int marks[3][5];
```




```
int marks[2][3]={90, 87, 78, 68, 62, 71};

int marks[2][3]={{90,87,78},{68, 62, 71}};

int marks[][3]={{90,87,78},{68, 62, 71}};
```

Ví dụ

```
#include <iostream>
using namespace std;
int main() {
  //Declare a 2D array with 3 rows, 4 columns
  int myArray[3][4];

  // Initialize the elements of the array
  for (int i = 0; i < 3; i++) {
      for (int j = 0; j < 4; j++) {
          myArray[i][j] = i + j;
      }
    }
```

```
// Print the elements of the array
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 4; j++) {
       cout << myArray[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Tham số mảng hai chiều cho hàm

- Cần cố định số cột của mảng hai chiều.
- Ví dụ:

```
void inputArray2D(int arr[][10], int num_row, int num_col) {
    for (int i = 0; i < num_row; i++) {
        for (int j = 0; j < num_col; j++) {
            cout << "Element[" << i << "][" << j << "]: ";
            cin >> arr[i][j];
        }
    }
}
```

```
int a[5][10];
inputArray2D(a, 3, 7);
```

```
//sum of 1-D array
int sumArr(int arr[], int n)
{
   int sum = 0;
   for(int i = 0; i < n; i++)
      sum += arr[i];
   return sum;
}
int main()
{
   int a[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};
   for(int row = 0; row < 3; row++)
   {
      int s = sumArr(a[row], 4); //sum of a row
      cout << “sum row: ” << row << “ is: ” << s << endl;
   }
}
```

Ma trận thưa

- Ma trận thưa (sparse matrix) là mà trận mà đa số các phần tử là 0.
- Cần tổ chức lưu trữ sao cho tiết kiệm nhưng thao tác được.

$$\begin{bmatrix} 1 & & & & \\ 5 & 3 & & & \\ 2 & 7 & -1 & & \\ 3 & 1 & 4 & 2 & \\ -9 & 2 & -8 & 1 & 7 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ & 3 & 6 & 7 & 8 \\ & & -1 & 9 & 1 \\ & & & 9 & 2 \\ & & & & 7 \end{bmatrix}$$

$$\begin{bmatrix} 4 & 1 & & & & \\ 5 & 1 & 2 & & & \\ & 9 & 3 & 1 & & \\ & & 4 & 2 & 2 & \\ & & & 5 & 1 & 9 \\ & & & & 8 & 7 \end{bmatrix}$$

Ma trận tam giác dưới

$$\begin{bmatrix} 1 & & & & \\ 5 & 3 & & & \\ 2 & 7 & -1 & & \\ 3 & 1 & 4 & 2 & \\ -9 & 2 & -8 & 1 & 7 \end{bmatrix}$$

$A_{i,j} = 0$ nếu $i < j$.

Dùng một mảng 1 chiều, đánh số theo dòng

B[] = {1, 5, 3, 2, 7, -1, 3, 1, 4, 2, -9, 2, -8, 1, 7}.

Khi đó:

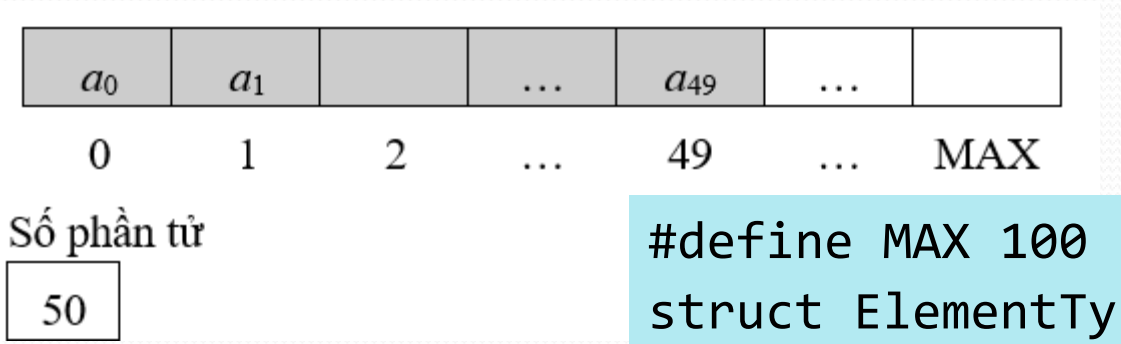
```
A[i][j] = 0, với i < j
A[i][j] = B[k], với k = (i * (i + 1))/2 + j, với i >= j
```

Ứng dụng

- Một số phạm vi ứng dụng của mảng gồm:
  - Dùng cài đặt các công cụ toán học như vector, ma trận, quan hệ,...
  - Dùng trong các cơ sở dữ liệu như mảng hai chiều các bản ghi.
  - Mảng dùng cài đặt nhiều cấu trúc dữ liệu thông dụng như: string, stack, queue, heap, hash table, ...
  - Mảng dùng để sắp xếp các phần tử theo thứ tự tăng hoặc giảm.
  - ...

Danh sách tổ chức bằng mảng

- Dùng một mảng để lưu các phần tử của danh sách.
- Dùng 1 ô nhớ lưu số phần tử hiện tại của danh sách.
- Ví dụ: $l = (a_0, a_1, \ldots, a_{49})$




```
#define MAX 100
struct ElementType{…};
struct ArrayList{
  ElementType arr[MAX];
  int         size;
};
```

```
#define MAX 100
struct Student
{
  string name;
  int    age;
  double gpa;
};
struct ListOfStudents{
  Student arr[MAX];
  int     size;
};
```

Thao tác trên danh sách

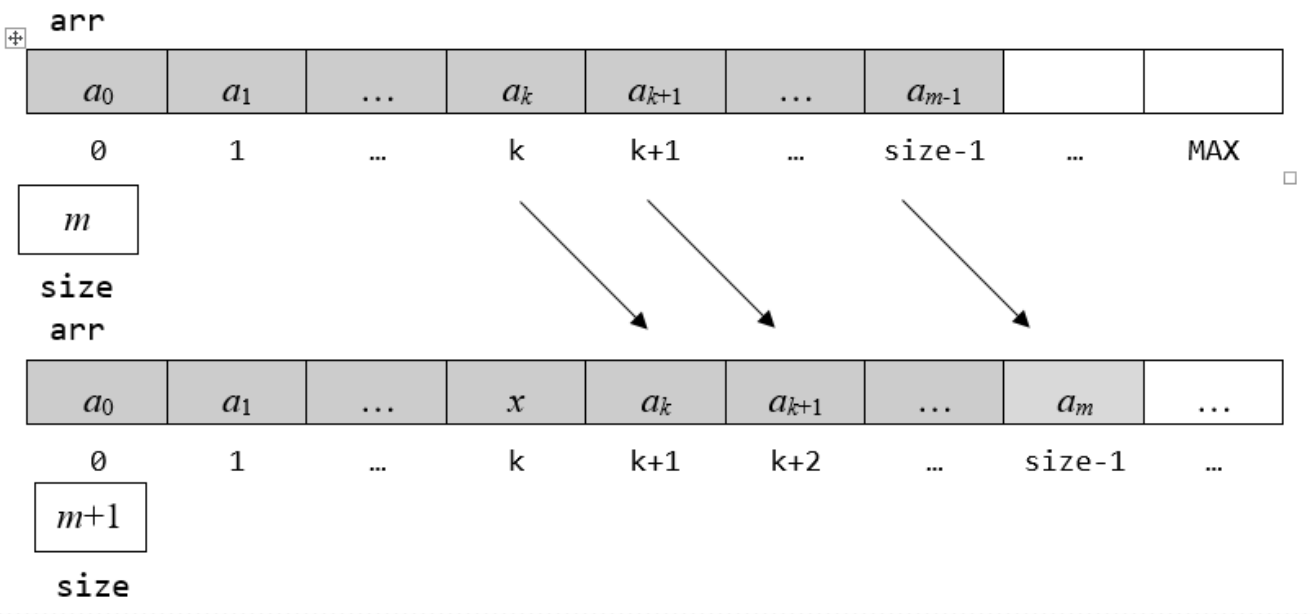
- Khởi tạo: gán số phần tử bằng 0

```
void initialize(ArrList &list)
{
 list.size = 0;
}
```

Khởi tạo danh sách sinh viên

```
void initialize(ListOfStudents &list)
{
 list.size = 0;
}
```

Thêm 1 phần tử vào danh sách

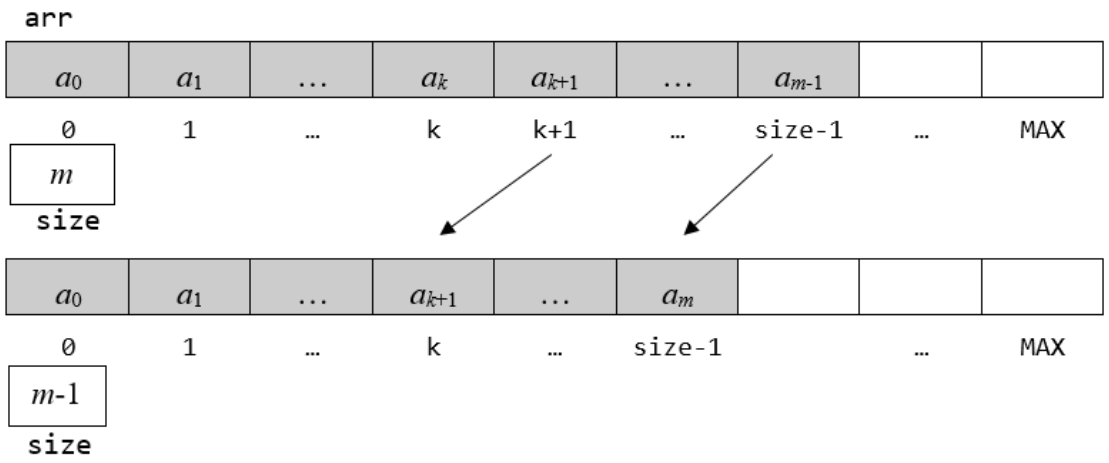


Input: Danh sách cần thêm list, phần tử cần thêm x, vị trí thêm k
Output: Danh sách list sau khi thêm
Action:
+ Chuyển các phần tử của mảng từ vị trí k đến vị trí (size-1) ra sau 1 vị trí.
+ Đưa phần tử x vào vị trí k của mảng
+ Tăng số phần tử của danh sách lên 1.

```
void insert(ListOfStudents &list, Student x, int k)
{
  if (list.size < MAX && k <= list.size)
  {
     for(int i = list.size - 1; i >= k; i--)
        list.arr[i+1] = list.arr[i];
     list.arr[k] = x;
     list.size++;
  }
}
```

```
void insert(ArrList &list, ElementType x, int k)
{
 if (list.size < MAX && k <= list.size)
 {
    for(int i = list.size - 1; i >= k; i--)
       list.arr[i+1] = list.arr[i];
    list.arr[k] = x;
    list.size++;
 }
}
```

Xóa 1 phần tử trong danh sách



Input: Danh sách cần xóa list, vị trí phần tử cần xóa k
Output: Danh sách list sau khi xóa
Action:
+ Chuyển các phần tử trong mảng từ vị trí sau k ra trước 1 vị trí.
+ Giảm số phần tử của danh sách đi 1.

```
void delete(ListOfStudents &list, int k)
{
  if (list.size > 0 && k < list.size)
  {
     for(int i = k+1; i <= list.size-1; i++)
        list.arr[i-1] = list.arr[i];
     list.size--;
  }
}
```

```
void delete(ArrList &list, int k)
{
  if (list.size > 0 && k < list.size)
  {
     for(int i = k+1; i <= list.size-1; i++)
        list.arr[i-1] = list.arr[i];
     list.size--;
  }
}
```

Tìm 1 phần tử trong danh sách

Input: Danh sách cần tìm list, khóa cần tìm x

Output: True nếu phần tử khóa x có trong danh sách, False nếu ngược lại

Action:

+ Duyệt lần lượt từng phần tử của danh sách:

    Nếu phần tử đang xét có khóa trùng với x thì trả về kết quả True.

+ Nếu hết danh sách thì trả về kết quả False.

```
bool search(ArrList list, KeyType x)
{
  for(int i = 0; i < list.size; i++)
     if (list.arr[i].key == x)
        return True;
  return False;
}
```

Sắp xếp danh sách

```
void BubbleSort(ArrList &list)
{
   int i, j; ElementType tmp;
   for (i = 1; i < list.size; i++)
      for (j = 1; j <= list.size-i; j++)
      if (list.arr[j].key < list.arr[j-1].key)
      {
        tmp = list.arr[j];
        list.arr[j] = list.arr[j-1];
        list.arr[j-1] = tmp;
      }
}
```

Nhận xét

- Ưu điểm: Truy xuất đến các phần tử trực tiếp. Do đó thuận lợi cho các thao tác cần truy xuất trực tiếp các phần tử như tìm nhị phân, quicksort.
- Nhược điểm:
  - Lãng phí ô nhớ vì có những ô nhớ còn trống do mảng được cấp phát 1 lần.
  - Các thao tác thêm và xóa phức tạp vì phải thay đổi nhiều phần tử trong danh sách.

Lớp vector trong C++

- #include <vector>
- Khai báo biến vector: vector<DataType> variable;
- Ví dụ:
  - vector<int> listOfInt;
  - vector<Student> listOfStudents;
- Tổ chức dữ liệu: vector sử dụng mảng động.
- Khi thêm tự động thêm ô nhớ khi gần hết ô nhớ
- Khi xóa tự động giảm ô nhớ nếu thừa nhiều

Phương thức

Phương thức	Chức năng	Độ phức tạp
`[index]`	Truy xuất đến phần tử tại vị trí index	O(1)
`size()`	Số phần tử trong danh sách	O(1)
`capacity()`	Kích thước bộ nhớ dùng để lưu trữ cho vector	O(1)
`empty()`	Cho biết vector có rỗng không	O(1)
`clear()`	Xóa các phần tử trong vector	O(1)
`push_back(e)`	Thêm một phần tử e vào cuối vector	O(1)
`insert(pos, e)`	Thêm một phần tử e vào vị trí pos	O(n)
`pop_back()`	Xóa phần tử cuối cùng của vector	O(1)
`erase(pos)`	Xóa phần tử tại vị trí pos trong vector	O(n)
`at(index)`	Trả về phần tử tại vị trí index trong vector	O(1)
`front()`	Trả về phần tử đầu tiên trong vector	O(1)
`back()`	Trả về phần tử cuối cùng trong vector	O(1)

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> numbers;

    // Adding elements to the vector
    numbers.push_back(10);
    numbers.push_back(20);
    numbers.push_back(30);
    numbers.push_back(40);

    // Printing the elements of the vector
    cout << "Elements of the vector: ";
    for (int i = 0; i < numbers.size(); i++) {
        cout << numbers[i] << " ";
    }
    cout << endl;
```

```cpp
//Removing the last element from the vector
   numbers.pop_back();

//Printing the updated vector
   cout<<"Elements of the updated vector:";
   for (int i = 0; i < numbers.size(); i++) {
        cout << numbers[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Bài tập

Bài 2.1 Trình bày thuật toán và cài đặt các thao tác sau trên danh sách sinh viên tổ chức bằng mảng:

a) Đếm số sinh viên có dtb>=x (x là tham số).

b) Xóa một sinh viên có họ tên x.

c) Tạo 1 danh sách mới chứa những sinh viên có dtb>=8.0.

d) Thêm 1 sinh viên x vào danh sách đã sắp thứ tự giảm của dtb.

e) Chèn một danh sách sinh viên l2 vào danh sách sinh viên l1 tại vị trí k.

f) Ghép 2 danh sách sinh viên đã sắp theo thứ tự giảm của dtb thành một danh sách mới được sắp theo thứ tự giảm của điểm trung bình.

g) Thống kê số lượng sinh viên theo tuổi trong danh sách sinh viên l.

Bài 2.2 Cho hai mảng số nguyên a có m phần tử, b có n phần tử. Hãy trình bày thuật toán và viết các hàm:

a) isSubArr(a, b): trả về kết quả true nếu b là một đoạn con của mảng a và false trong trường hợp ngược lại.

b) isSubSequence(a, b): trả về kết quả true nếu b là dãy con của a và false trong trường hợp ngược lại.

Ví dụ: a = (1, 5, 2, 1, 3, 7, 2), b = (5, 1, 2), c = (1, 2, 5, 3), d = (2, 1, 3) thì b là dãy con của a, c không phải là dãy con của a, d là đoạn con của a.

Bài 2.3 Sử dụng mảng hai chiều để lưu điểm m môn học của n học sinh. Viết các hàm thực hiện:

a) Tính trung bình cộng điểm của từng học sinh

b) Tìm điểm cao nhất của từng môn học

c) Sắp xếp danh sách học sinh theo thứ tự giảm của điểm trung bình cộng