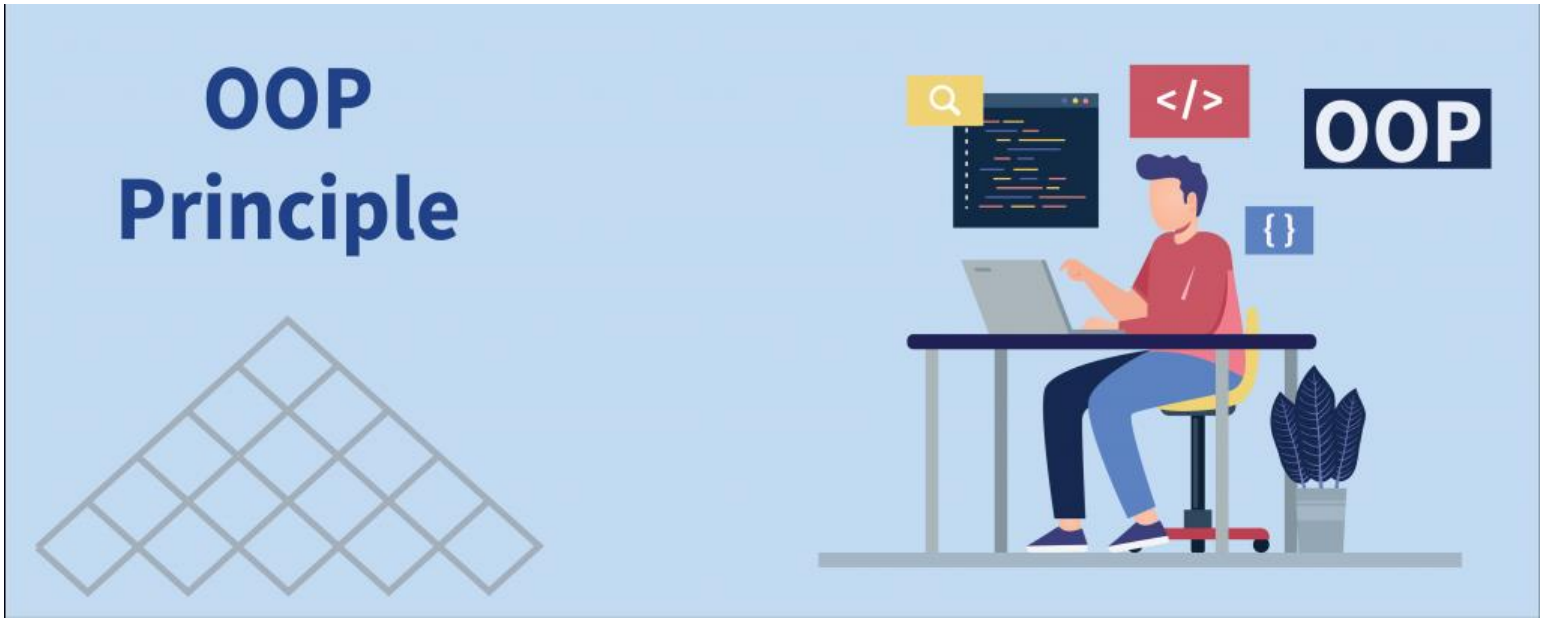


NHẬP XUẤT FILE

Ths. Đinh Thị Thúy – AI LAB/TLU
Thuydt@thanglong.edu.vn – PAI006

Đọc ghi file văn bản

- Tạo dòng, nối dòng tới file
- Dùng dòng và toán tử nhập xuất để đọc, ghi file
- Khai báo các chế độ mở file
- Kiểm tra các cờ lỗi

Dòng: Tạo dòng, nối dòng tới file

- C++ thiết kế những lớp giúp xử lý file - gọi là các dòng (ống)
- Lập trình viên chỉ cần tạo dòng phù hợp là làm việc được với file
  - Dòng để đọc file = biến thuộc lớp ifstream
  - Dòng để ghi file = biến thuộc lớp ofstream
  - Dòng hai chiều, có thể cả đọc cả ghi = biến thuộc lớp fstream
  - Các lớp trên đều có trong thư viện <fstream>
- Sau khi tạo dòng, cần nối nó với file muốn xử lý:

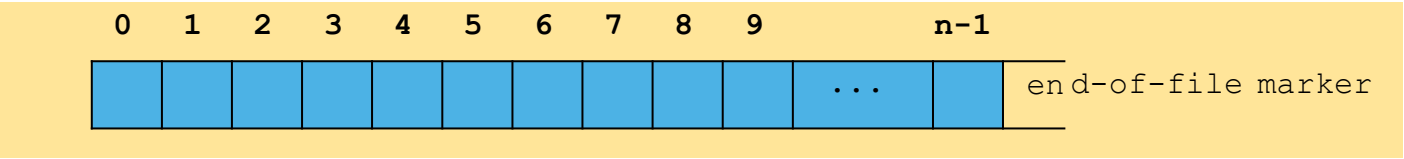
```
// tên file để trong nháy kép
ofstream out ("output.txt");    // nối dòng tới file qua hàm tạo
istream in;
in.open("input.txt");           // hoặc qua hàm open
```

- Nếu nối thành công thì dòng nhận giá trị true, nếu không sẽ nhận false

```
if (!out)  cout << "Loi \n" ;      // không nối được thì không thể làm gì
else  {…}                          // nối được mới xử lý file tiếp
```

Dòng và việc đọc ghi file

- File như nguồn nước, ở cuối nguồn là ký tự đặc biệt kết file (EOF)
- Dòng nối tới nguồn và sẽ vớt dữ liệu hay đổ dữ liệu vào nguồn



- Dòng đọc vớt dữ liệu từ file hệt cách cin vớt dữ liệu từ bàn phím
  - fin >> n; // đọc dữ liệu đầu tiên khác trắng
  - fin >> int1 >> char2 >> bool3; // đọc kiểu lợp ngói
  - getline(fin, s) // đọc từ đầu đến khi gặp enter
  - fin.ignore(); fin.ignore(100, '\n'); // nhảy qua
  - fin.get (c) // đọc 1 ký tự
  - if (fin >> a) {…} // kiểm tra liệu việc đọc ổn thoả

Dòng và việc đọc ghi file

- Dòng ghi đổ dữ liệu ra file hệt cách cout đổ dữ liệu ra màn hình:
  - out << n; // in ra một số
  - out << int1 << " " << string1 << endl; // in kiểu lợp ngói
  - out << setprecision(2) << fixed << float1; // định dạng
  - out << setw (8) << int1 << '\n'; // giống hàng
- Dòng fstream sẽ đọc hay ghi dữ liệu tuỳ vào toán tử kết hợp
  - f >> n; // toán tử >> để đọc
  - f << "Hello" << endl; // toán tử << để ghi
- Khi làm việc xong cần gỡ dòng khỏi file, còn gọi là đóng dòng:
  - fin.close(); // các dòng fin, out, f vẫn tồn tại, chỉ thôi nối với file
  - out.close();
  - f.close();

Tổng kết quy trình đọc và ghi file

1) Thêm thư viện <fstream>
2) Khai báo dòng phù hợp (istream/ostream/fstream) và nối tới file

```
ostream out ("data.txt");     // truyền tên file vào hàm tạo
```

hoặc

```
ostream out;
out.open("data.txt");          // truyền tên file vào hàm open
```

3) Kiểm tra dòng có nối thành công

```
if (!out)  cout << "Loi \n";   // không nối được thì không thể làm gì
else  {…}                      // nối được mới xử lý file tiếp
```

4) Tiến hành xử lý file:

```
in >> n; …
   out << n << endl; …
```

5) Xong việc thì đóng kết nối

```
in.close(); out.close();
```

VD: Tạo dòng, nối dòng, kiểm tra

Hãy tạo các dòng nối tới các file sau, tuỳ vào nối thành công hay không mà in ra thông báo tương ứng

- Dòng để đọc file Sinhvien.txt
- Dòng để ghi vào file Diemso.txt
- Dòng để vừa đọc vừa ghi file InputOutput.txt

Sau đó đóng các file lại

Hãy test trong 3 trường hợp trên, khi nào bắt buộc phải tạo file rồi mới được nối dòng tới nó, trường hợp nào không cần? Và trong trường hợp được phép chưa tạo file vẫn nối dòng tới nó thì điều gì xảy ra?

VD: Đọc, ghi file

a) Cho file random.txt chứa các số nguyên, hãy đọc file này và tính:
   - Số lượng số, tổng các số, giá trị trung bình các số trong file

b) Ghi các thông tin trên ra file result.txt

c) Sửa bài câu 2 để thay vì dùng tên file random.txt và result.txt thì tên file đọc và ghi sẽ nhập từ bàn phím.

VD: Đọc file

```
// File random.txt chứa các số ngẫu nhiên. Đọc file này, rồi hiển thị:
// Số lượng, tổng và giá trị trung bình của các số trong file
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ifstream fin("random.txt");

    if (!fin)
        cout << "Loi: khong doc duoc file \n";
    else {
        double n, tong = 0;
        int dem = 0;
        while (fin >> n) //còn đọc được thì còn làm tiếp;
        {
            dem ++;
            tong += n;
        }
        cout << "Tong: " << tong << endl
            << "So luong: " << dem << endl;
        if (dem)
            cout  << "Trung binh: " << tong/dem << endl;
        else
            cout << "Khong tinh duoc trung binh \n";
    }
    fin.close();

    return 0;
}
```

Tạo ống fin và nối nó với nguồn random.txt

Nếu nối ống không được, fin sẽ là false, ngược lại là true

fin >> n đọc dữ liệu file vào biến n, nó cũng trả về true nếu đọc thành công, và false nếu không đọc được (do hết file, dữ liệu sai kiểu, …)

Dùng xong thì gỡ ống khỏi nguồn. Ông fin lúc này vẫn tồn tại, chỉ đang không nối tới nguồn nào

VD: Đọc & ghi file

```
ifstream fin("random.txt");

if (!fin)
    cout << "Loi: khong doc duoc file \n";
else {
    double n, tong = 0;
    int dem = 0;
    while (fin >> n) //còn đọc được thì còn làm tiếp;
    {
        dem ++;
        tong += n;
    }
    ofstream fout("result.txt");
    fout << "Tong: " << tong << endl
        << "So luong: " << dem << endl;
    if (dem)
        fout  << "Trung binh: " << tong/dem << endl;
    else
        fout << "Khong tinh duoc trung binh \n";
    fout.close();
}
fin.close();

return 0;
```

VD: Đọc & ghi với tên file nhập vào

```
string finName, foutName;       // tên file sẽ đọc, tên file sẽ ghi
cout << "Ten file de doc: ";
cin >> finName;
cout << "Ten file de ghi:";
cin >> foutName;
ifstream fin(finName.c_str());  // tuỳ trình biên dịch, có lúc cần
                                // chuyển tên file thành c-string

if (!fin)
    cout << "Loi, khong doc duoc file" << endl;
else {
    double n, tong = 0;
    int dem = 0;
    while (fin >> n)            //còn đọc được thì còn làm tiếp;
    {
        dem ++;
        tong += n;
    }
    ofstream fout(foutName.c_str()); //khai báo dòng để ghi file
    if (!fout)
        cout << "Loi: khong ghi duoc file \n" ;
    else{
        fout << "Tong cac so: " << tong << endl
        << "So cac so: " << dem << endl;
        if (dem)
            fout  << "Trung binh cac so: " << tong/dem << endl;
        else
            fout << "Khong tinh duoc trung binh \n";
    }
    fout.close();
}
fin.close();
```

Các hằng mở file

- Bổ sung các tuỳ chỉnh dòng tinh vi hơn
- Được đặt ở tham số thứ hai của hàm tạo hay hàm open:

```
fstream file1 ("fileOut.dat",ios::out);
fstream file2;
file2.open("fileIn.dat",ios::in);
```

  - ios::in, ios::out, .. – là các chế độ mở file, còn gọi là các cờ file
- Có thể kết hợp các chế độ với nhau

```
fstream file3 ("file.dat", ios::in | ios::out);
```

  - Cần kết hợp hợp lý, không phải mọi thứ đều đi được với nhau
    - VD: Dòng ofstream không thể đi với ios::in

Hằng mặc định

- Dòng istream:
  - Mặc định đi với ios::in
  - Chỉ mở để đọc, không thể ghi
  - Thường báo lỗi nếu file chưa tồn tại
- Dòng ostream:
  - Mặc định đi với ios::out
  - Chỉ mở để ghi, không thể đọc
  - Nếu file chưa tồn tại thì tạo file mới với tên đó
  - Nếu file tồn tại thì xoá nội dung cũ và ghi lại từ đầu file

Danh sách các hằng mở file

ios::app	Tạo file mới nếu chưa có, nếu có thì chỉ ghi nối vào sau file đã có
ios::ate	Bắt đầu bằng nhảy đến cuối file đã có; nhưng nếu muốn có thể ghi ở mọi chỗ
ios::binary	Đọc/ghi ở chế độ nhị phân (không phải chế độ văn bản)
ios::in	Mở file để đọc
ios::out	Mở file để ghi

Các loại cờ lỗi

- Bổ sung các kiểm tra lỗi tinh vi hơn
- Dòng có các biến bool ghi lại trạng thái thành công/thất bại của các thao tác – còn gọi là các cờ lỗi (error flag/error state bit)
  - Một thao tác thất bại sẽ bật một hay vài cờ liên quan thao tác đó
- Dòng cung cấp hàm thành viên để đọc giá trị các cờ này
- Từ đó, có thể ”bắt mạch” được lỗi do thao tác nào sinh ra và nên xử lý tiếp ra sao

Các loại cờ lỗi

ios::eofbit	Cờ này bật lên khi gặp cuối file
ios::failbit	Bật khi thao tác vừa làm thất bại
ios::hardfail	Bật khi xuất hiện lỗi không thể hồi phục
ios::badbit	Bật khi thử thực hiện một thao tác không hợp lệ (dù chưa kịp làm xong)
ios::goodbit	Bật khi mọi cờ còn lại đều tắt

Các hàm kiểm tra cờ lỗi

`eof()`	Trả về true nếu bật cờ **`eofbit`**, false nếu ngược lại
`fail()`	Trả về true nếu bật **`failbit`** hoặc **`hardfail`**, false nếu ngược lại
`bad()`	Trả về true nếu bật **`badbit`**, false nếu ngược lại
`good()`	Trả về true nếu bật **`goodbit`**, false nếu ngược lại
`clear()`	Xoá mọi cờ về false (nếu không tham số), hoặc truyền vào một cờ cụ thể để xoá

Mở rộng: Các hàm tiện ích khi đọc file

- Hoàn toàn tương tự các hàm tiên ích khi đọc bàn phím:
  - cin >> s : Nhảy qua dấu trắng, để lại mọi thứ còn lại trong nguồn
  - getline (cin, s, delim): Vớt cả dấu trắng, dừng khi gặp delim. Không vớt vẫn xoá delim khỏi nguồn, để lại mọi thứ sau delim.
  - Nếu cin trước getline: cin để lại enter mà getline lại vớt được enter, nên lệnh getline hoàn thành và sẽ không đọc gì thêm
    - Dùng cin.ignore() rồi mới gọi getline
  - cin.get (c): Vớt 1 ký tự đầu tiên, kể cả dấu trắng, để lại mọi thứ từ đó. VD nhập enter thì chương trình thu được c = '\n'

Mở rộng: Các hàm tiện ích khi ghi file

- Hoàn toàn tương tự các hàm tiên ích khi in ra màn hình:
  - out << setw(n) : in ra file với độ rộng tối thiểu là n, căn phải
  - out << setprecision(n) << fixed : in ra file với n chữ số sau phẩy
  - out << left: in ra file căn trái
  - out.put(c) : in ra c như một ký tự (out.put(65) khác out << 65)

VD: Copy file nhờ copy từng ký tự

```
char ch;
  while (infile.get(ch))
     outfile.put(ch);
  infile.close();
  outfile.close();
```

Đọc ghi file nhị phân

- Khái niệm file văn bản vs. file nhị phân
- Dùng hàm read, write với biến đơn, với mảng, với bản ghi, với mảng bản ghi

File văn bản vs. file nhị phân

- Mặc định file sẽ được mở theo chế độ dữ liệu dạng văn bản (text file)
  - Ưu điểm của text file là dễ hiểu, What You See Is What You Get
  - Nhưng ai cũng đọc được nội dung file, bảo mật kém
  - Chỉ đọc ghi từng biến cơ bản, không thể đọc/ghi cả khối dữ liệu
- File nhị phân (binary file) lưu dữ liệu dạng nhị phân như trong máy
  - Ai mở nội dung file đọc cũng không hiểu
  - Cho phép đọc, ghi cả khối dữ liệu (cả một đối tượng, cả một mảng)
- Để mở file ở chế độ nhị phân: truyền thêm hằng ios::binary

```
ifstream in("inFile.dat", ios::binary);
ofstream out ("outFile.dat", ios::binary);
```

- Đọc, ghi file kiểu nhị phân dùng hàm read, write thay vì >>, <<

Hàm read, write

```
read(char *buffer, int numberBytes)
write(char *buffer,int numberBytes)
```

- numberBytes: số lượng byte trao đổi giữa bộ nhớ và file
- buffer: địa chỉ bộ nhớ mà tại đó sẽ chứa chuỗi byte trao đổi với file
  - Phải là địa chỉ kiểu char (tức là char *), nếu khác thì cần ép kiểu reinterpret_cast <char * > (địa_chỉ) // ép về địa chỉ kiểu char

```
ofstream out("file.dat",ios::binary);
int n = 3;                  out.write(reinterpret_cast<char *> (&n), sizeof(n));
out.close();

ifstream in("file.dat", ios::binary);
int n1;                     in.read(reinterpret_cast<char *> (&n1), sizeof(n1));
in.close();
cout << n1 << endl;                                                    3
```

VD: Hàm read và write

```
read(char *buffer, int numberBytes)
write(char *buffer,int numberBytes)
```

a) Mở file.dat ở dạng nhị phân, ghi ra các thông tin sau rồi đóng file:
   - Một số nguyên 3;
   - Một số thực bằng 1.2
   - Một giá trị bool là false
   - Một ký tự 'C'
   - Một mảng 2 phần tử số thực, đã khởi tạo giá trị nào đó

b) Mở lại file.dat ở dạng nhị phân để lần lượt đọc các thông tin sau:
   - Đọc từ file vào một số nguyên, rồi số thực, rồi số bool, rồi ký tự, rồi mảng số thực 2 phần tử
   - In ra các giá trị đọc được để kiểm tra

c) Thay đoạn mã mảng 2 phần tử thành mảng 10 phần tử, song từ file sẽ chỉ đọc vào 2 phần tử đầu mảng

VD: Hàm write

```
// mở file ghi ở chế độ nhị phân
ofstream out("file.dat",ios::binary);

// ghi ra lần lượt từng biến
int n = 3;          out.write(reinterpret_cast<char *> (&n), sizeof(n));
double d = 1.2;     out.write (reinterpret_cast<char *> (&d), sizeof(d));
bool b = false;     out.write (reinterpret_cast<char *> (&b), sizeof(b));
char c = 'C';       out.write (&c,sizeof(c)); // tại sao ko cần ép kiểu?


// ghi ra cả mảng
double a[2] = { 12.3, 34.5 };
out.write(reinterpret_cast<char *>(a),sizeof(a)); // tại sao không cần & trước d ?


// đóng file ghi
out.close();
```

VD: Hàm read

```
// mở file đọc ở chế độ nhị phân
ifstream in("file.dat", ios::binary);

// đọc vào lần lượt từng biến
int n1;             in.read(reinterpret_cast<char *> (&n1), sizeof(n1));
double d1;          in.read (reinterpret_cast<char *> (&d1), sizeof(d1));
bool b1;            in.read (reinterpret_cast<char *> (&b1), sizeof(b1));
char c1;            in.read (&c1,sizeof(c1)); // tại sao ko cần ép kiểu?

// nếu đọc đủ hết mảng
double a1[2];
in.read(reinterpret_cast<char *>(a1),sizeof(a)); // tại sao không cần & trước d ?
cout << n1 << " " << d1 << " " << b1 << " " << endl;
cout << a1[0] << " " << a1[1] << endl;
```

```
3 1.2 0
12.3 34.5
```

```
// nếu đọc vào chỉ một phần mảng
const int SIZE = 10;
double data[SIZE];
in.read(reinterpret_cast<char *>(data),  2 * sizeof(double));
// chỉ data[0] và data[1] chứa dữ liệu vì chỉ đọc 2 biến double.
for (int i = 0; i < SIZE; i++)
    cout << data[i] << " ";
cout << endl;

// đóng file đọc
in.close();
```

```
3 1.2 0
12.3 34.5 0 0 0 0 0 0 0 0
```

read và write bản ghi

- Một bản ghi là một tập hợp các biến đi cùng nhau mang một ý nghĩa nào đó, thường hiểu là biến kiểu struct hay kiểu lớp
- Trong ngữ cảnh đọc ghi file của C++, ta giới hạn bản ghi chỉ là các biến struct, vì struct cho phép truy cập dữ liệu không cần getter, setter.
- Đọc ghi kiểu nhị phân cả một bản ghi

```
read(char *buffer, int numberBytes)
write(char *buffer,int numberBytes)
```

  - Vẫn chỉ cần địa chỉ biến (lấy từ &) và kích thước biến (lấy từ sizeof)
  - Mở rộng đọc, ghi mảng thì cần truyền địa chỉ mảng (chính là tên mảng) và kích thước mảng (cũng lấy từ sizeof(tên mảng))
  - Luôn nhớ ép kiểu địa chỉ sang char*

VD: read và write cả một bản ghi

```
struct SinhVien{
    char ten[30]; // để đọc ghi nhị phân thì xâu để kiểu mảng ký tự thay vì kiểu string
    int namsinh;
    float diemtrungbinh;
};


SinhVien sv1 = {"Nguyen Van Nam", 2002, 7.9}, sv2;

ofstream out ("sinhvien.dat", ios::binary);
out.write(reinterpret_cast<char *> (&sv1), sizeof(sv1));
out.close();

ifstream in ("sinhvien.dat", ios::binary);
in.read(reinterpret_cast<char *> (&sv2), sizeof(sv2));
in.close();

cout << sv2.ten << endl << sv2.namsinh << endl << sv2.diemtrungbinh << endl;

return 0;
```

```
Nguyen Van Nam
2002
7.9
```

Lưu ý khi read và write một bản ghi

- Khi ghi vào file nhị phân, biến struct không được phép chứa con trỏ
- Kiểu string thực chất bên trong lại chứa một con trỏ cấp phát động, nên biến struct không được chứa các biến thành viên kiểu string nếu muốn ghi vào file nhị phân
- Giải pháp: biểu diễn xâu bằng kiểu mảng ký tự thay vì kiểu string.
  - VD: char name [30];

VD: read và write một bản ghi

- Viết cấu trúc Sach chứa các thông tin:
  - Tên sách, tên tác giả, năm xuất bản, số trang
- Main tạo ra một cuốn sách b1 với thông tin tự chọn, mở file sinhvien.dat dạng ghi ở chế độ nhị phân, ghi b1 vào file,
- Sau đó lại mở file này dạng đọc ở chế độ nhị phân, tạo một cuốn sách b2 khác, đọc từ file vào biến này, rồi in ra mọi thông tin biến đó, kiểm tra có trùng b1.

VD: read và write mảng bản ghi

Vẫn trên cấu trúc Sach, viết main tạo ra một mảng 10 cuốn sách, tự khởi tạo tuỳ ý

a) Ghi cả mảng này vào file sach.dat kiểu nhị phân
b) Lại mở sach.dat để đọc kiểu nhị phân, tạo một mảng Sach nữa cũng 10 phần tử, đọc file vào mảng đó, in ra thông tin mọi cuốn sách, kiểm tra có giống mảng ban đầu
c) Nhập vào số n, tạo mảng n cuốn sách, nhập thông tin cho mảng từ bàn phím, ghi cả mảng vào file sach2.dat kiểu nhị phân
d) Mở sach2.dat để đọc kiểu nhị phân, tạo một mảng Sach nữa có n phần tử, đọc file vào mảng đó, in ra mọi cuốn sách cùng số lượng sách xuất bản 2022

VD: read và write mảng bản ghi

Doanh số chi nhánh

- Viết struct CorpSale lưu các dữ liệu sau của một chi nhánh công ty:
  - Tên chi nhánh (chẳng hạn như Đông, Tây, Bắc hoặc Nam)
  - Doanh số 4 quý
- Tạo 4 chi nhánh tên Đông, Tây, Nam, Bắc, nhập nốt dữ liệu còn lại từ bàn phím. Sau đó ghi dữ liệu mọi chi nhánh vào một tệp.
- Xác thực đầu vào: Không chấp nhận số liệu bán hàng âm

Truy cập ngẫu nhiên

- Khái niệm truy cập tuần tự vs. truy cập ngẫu nhiên
- Các hàm hỗ trợ truy cập ngẫu nhiên: seekg, seekp

Truy cập tuần tự và truy cập ngẫu nhiên

- Truy cập tuần tự: truy cập lần lượt các vị trí từ đầu file đến cuối file
  - Để đọc/ghi bản ghi 100, bắt buộc phải đọc/ghi qua 99 bản ghi trước đó
- Truy cập ngẫu nhiên: trực tiếp nhảy đến vị trí mà ta muốn làm việc
  - Để đọc/ghi bản ghi 100, chỉ cần chỉnh đầu đọc/ghi nhảy đến đó.
- Truy cập ngẫu nhiên cần hàm giúp nhảy đến vị trí cho trước
  - File đọc: dùng hàm seekg
  - File ghi: dùng hàm seekp
  - File đọc lẫn ghi: dùng được cả 2 hàm trên và như nhau

seekg(), seekp() - nhảy đến vị trí cho trước

```
seekg(offset, place) // dùng bởi file đọc
seekp(offset, place) // dùng bởi file ghi
//file đọc lẫn ghi gọi được cả 2 hàm và như nhau
```

- Ý nghĩa: di chuyển một khoảng **offset** sau vị trí **place**
  - **offset**: số byte cần di chuyển tính từ **place**, là kiểu **long**
  - **place**: vị trí trong file mà từ đó bắt đầu di chuyển
    - **ios::beg**: đầu file
    - **ios::end**: cuối file
    - **ios::cur**: vị trí hiện tại của đầu đọc/ghi
- Về đầu file, cuối file, về sau vị trí hiện tại n byte:

```
in.seekg(0, ios::beg);      out.seekp(0, ios::beg);
in.seekg(0, ios::end);      out.seekp(0, ios::end);
in.seekg(n, ios::cur);      out.seekp(0, ios::cur);
```

VD: seekg(), seekp() - hàm truy cập ngẫu nhiên

```
// di chuyển đầu đọc đến vị trí 25 byte
// tính từ đầu file
inData.seekg(25L, ios::beg);

// di chuyển đầu ghi đến vị trí 10 byte
// trước vị trí hiện thời của đầu ghi
outData.seekp(-10L, ios::cur);
```

Dùng seekg(), seekp() đi ngược về trước vị trí hiện tại

- Thường xuyên nhất là dùng seekg() đi về điểm đầu file để đọc lại
  - Khi đọc hết file, nếu muốn đọc lần nữa cần di chuyển về đầu file
  - Song khi hết file, cờ eof đã bị bật và sẽ chặn mọi thao táo sau đó, nếu muốn gọi thao tác về đầu file, trước tiên phải xoá mọi cờ lỗi;

```
inFile.clear();
inFile.seekg(0, ios::beg);
```

- Có lúc ta cũng dùng seekp đi về đằng trước (đầu file hoặc đâu đó) để ghi tiếp, song điều gì sẽ xảy ra: ghi đè lên hay ghi chèn thêm (??)

VD: seekg(), seekp() - hàm truy cập ngẫu nhiên

Tail Program

- Viết chương trình yêu cầu người dùng nhập tên tệp. Chương trình sẽ hiển thị 10 dòng cuối cùng của tệp trên màn hình (“đuôi” của tệp). Nếu tệp có ít hơn 10 dòng, toàn bộ tệp sẽ được hiển thị, với thông báo cho biết toàn bộ tệp đã được hiển thị.
- Chú ý: Bằng trình soạn thảo, tạo một tệp văn bản đơn giản có thể kiểm tra được chương trình này.

Tổng kết

- Khai báo dòng và nối nó tới file
- Làm việc ở chế độ file văn bản dùng dòng và <<, >>
- Làm việc ở chế độ file nhị phân dung dòng và read, write
- Truy cập file kiểu ngẫu nhiên với seekg, seekp

Keep calm and carry on