# Bài 12: **Thư viện STL** (***Standard Template Library***)

# Khái niệm

- STL là thư viện chuẩn của C++, được xây dựng sẵn
- Cài đặt các cấu trúc dữ liệu và thuật toán thông dụng
- Bao gồm các lớp và hàm khuôn mẫu, cho phép làm việc với dữ liệu tổng quát
- Nằm trong một namespace có tên `std`
- Các phần chính:
  - Các lớp dữ liệu cơ bản: `string`, `complex`
  - Xuất nhập (IO)
  - Các lớp chứa (containers): `list`, `vector`, `deque`, `stack`, `map`, `set`,...
  - Duyệt phần tử của các lớp chứa (iterators)
  - Một số thuật toán thông dụng: tìm kiếm, so sánh, sắp xếp,…
  - Quản lý bộ nhớ, con trỏ
  - Xử lý ngoại lệ (exception handling)

# Xử lý chuỗi

- `#include <string>`
- Lớp `string` cho chuỗi ASCII và `wstring` cho Unicode
- Các thao tác cơ bản: `+`, `+=` (nối chuỗi); `==`, `!=`, `>`, `<`, `>=`, `<=` (so sánh); `<<` (xuất), `>>` (nhập)
- Độ dài chuỗi: `int string::length() const`
- Chuỗi con:
  `string string::substr(int off, int count) const`
- Tìm chuỗi con:
  `int string::find(const char* str, int pos) const`
- Đổi sang chuỗi của C: `const char* string::c_str() const`
- Đổi sang số và ngược lại (C++11):

```
[int|long|float|double] sto[i|l|f|d](const string& s);
string to_string([int|long|float|double] n);
wstring to_wstring([int|long|float|double] n);
```

# Xử lý chuỗi: Ví dụ

```
string s1, s2("test123");
cin >> s1;
s1 += to_string(123);
cout << (s2==s1 ? "same" :
            "different") << endl;

int pos = s2.find("est");
string s3 = s2.substr(pos, 4);

char s4[100];
strcpy(s4, s3.c_str());
cout << s4 << endl;
```

# Các lớp chứa (*containers*)

# Mảng: vector

- Là mảng động
- Có thể chứa dữ liệu kiểu bất kỳ (template): `vector<type>`
- `#include <vector>`
- Ví dụ sử dụng:

```
int p[] = {4, 2, 6};
vector<int> a(p, p+3);        // khởi tạo từ mảng C
a.push_back(1);               // thêm vào cuối
a.insert(a.begin() + 2, 3); // thêm ở vị trí 2
a.insert(a.end() - 1, 5);   // thêm ở vị trí 1 từ cuối
a[3] = 10;                    // phần tử thứ 4

vector<int>::iterator i;     // duyệt xuôi
for (i = a.begin(); i != a.end(); i++) *i += 5;

vector<int>::reverse_iterator j; // duyệt ngược
for (j = a.rbegin(); j != a.rend(); j++)
   cout << *j << ' ';
```

# iterator

- Các lớp chứa của STL (vector, list,…) có định nghĩa kiểu `iterator` tương ứng để duyệt các phần tử (theo thứ tự xuôi)
  - Mỗi `iterator` chứa vị trí của một phần tử
  - Các hàm `begin()` và `end()` trả về một `iterator` tương ứng với các vị trí đầu và cuối
  - Các toán tử với iterator:

| | |
|---|---|
| `i++` | phần tử kế tiếp |
| `i--` | phần tử liền trước |
| `*i` | giá trị của phần tử |
| `i->` | lấy thành phần của phần tử |

- Tương tự, có `reverse_iterator` để duyệt theo thứ tự ngược
  - Các hàm `rbegin()` và `rend()`

# Danh sách liên kết: list

- Có thể chứa dữ liệu kiểu bất kỳ (template): `list<type>`
- `#include <list>`
- Duyệt danh sách dùng `iterator` tương tự như với `vector`
- Ví dụ sử dụng:

```
double p[] = {1.2, 0.7, 2.2, 3.21, 6.4};
list<double> l(p, p+5); // khởi tạo từ mảng C
l.push_back(3.4);       // thêm vào cuối
l.pop_front();          // xoá phần tử đầu

list<double>::iterator i = l.begin();  // phần tử đầu
*i = 4.122;             // gán giá trị
i++;                    // phần tử kế tiếp
l.insert(i, 5.0); // chèn phần tử
l.erase(i);             // xoá phần tử
l.sort();               // sắp xếp (tăng dần)
for (i = l.begin(); i != l.end(); i++) // duyệt xuôi
   cout << *i << ' ';
```

# Thuật toán: tìm kiếm

- Phần tử lớn nhất, bé nhất:
  - `vector<float>::iterator p = max_element(a.begin()+2, a.end()-3);`
  - `list<string>::iterator p = min_element(l.begin(), l.end());`
  - Dựa trên các toán tử so sánh → cần định nghĩa nếu chưa có
- Tìm đúng giá trị:
  - `list<float>::iterator p = find(p1, p2, 2.5f);`
- Tìm theo tiêu chuẩn: cần định nghĩa một hàm đánh giá
  - `bool isOdd(int i) { return i%2 == 1; }`

    `list<int>::iterator p = find_if(p1, p2, isOdd);`
- Tìm kiếm và thay thế, xoá:
  - `replace_if(p1, p2, isOdd, 10);`
  - `remove_if(p1, p2, isOdd);`

# Thuật toán: sắp xếp

- Sắp xếp mảng:
  - Dùng toán tử so sánh:
    - `sort(a.begin(), a.end());`
    - Phải định nghĩa toán tử "<" cho kiểu dữ liệu được chứa
  - Dùng hàm so sánh tự định nghĩa:

```
bool compare(const table& a, const table& b) {
    return a.c1 < b.c1 ||
        (a.c1 == b.c1 && a.c2 < b.c2);
}
sort(a.begin(), a.end(), compare);
```

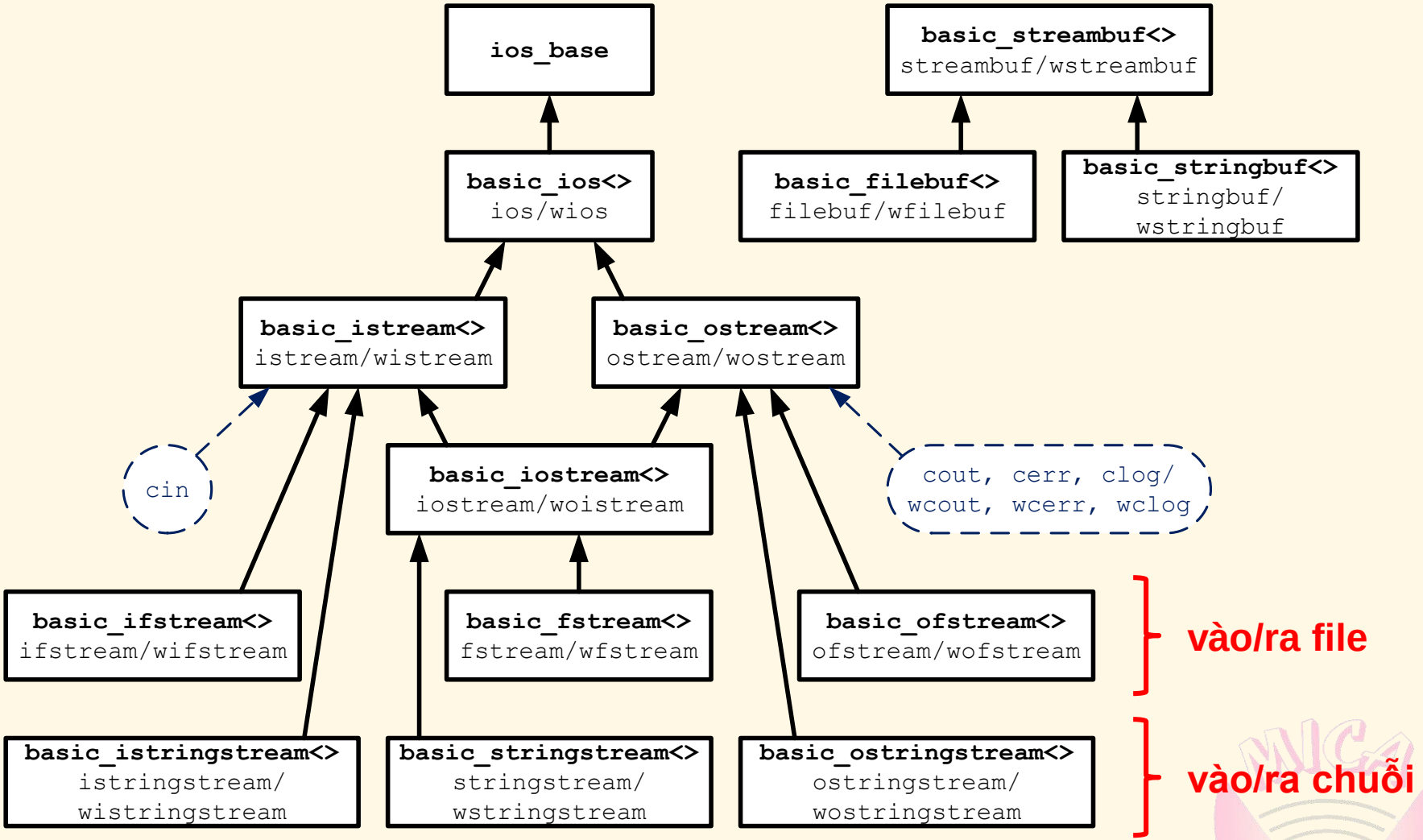
- Sắp xếp danh sách:
  - `l.sort();`
  - `l.sort(compare);`

# Xuất/nhập (*input/output*)

# Tổng quan

- Trong STL, việc xuất nhập được thông qua các "luồng thông tin" (data streams)
  - `#include <iostream>`
  - Luồng xuất (output streams): để xuất dữ liệu
    - Toán tử <<
    - Lớp cơ sở: `basic_istream<type>`
  - Luồng nhập (input streams): để nhập dữ liệu
    - Toán tử >>
    - Lớp cơ sở: `basic_ostream<type>`
  - Luồng xuất nhập (input/output streams): cả xuất và nhập
    - Toán tử >> và <<
    - Lớp cơ sở: `basic_iostream<type>`
- Mỗi kiểu đối tượng muốn làm việc được với các lớp trên cần phải được định nghĩa toán tử >> và <<

# Sơ đồ các lớp xuất/nhập



# Vào/ra chuẩn

- Các đối tượng
  - `cin` thuộc kiểu `istream`, tương đương với `stdin`
  - `cout` thuộc kiểu `ostream`, tương đương với `stdout`
  - `cerr` thuộc kiểu `ostream`, tương đương với `stderr`
  - Các đối tượng `wcin`, `wcout`, `wcerr` để làm việc với Unicode
- Ví dụ:

```
int i;
float a[10];
cout << "Nhap i va a[i]: ";
if (cin >> i >> a[i]) {
   cout <<  "a[" << i << "] = " << a[i] << endl;
   cout.flush();
} else cerr << "Nhap du lieu loi" << endl;
```

- Đọc một dòng:

```
string s;
getline(cin, s);
```

- Chú ý tránh sử dụng lẫn lộn với các hàm của C

# Định dạng dữ liệu xuất

- Các hàm thay đổi định dạng:
  - `setf(fmtflags flag, fmtflags mask)`: thay đổi các cờ định dạng
    - `dec/hex/oct`: số nguyên hệ cơ số 10/16/8 (`basefield`)
    - `fixed/scientific`: số thực thập phân hoặc khoa học (`floatfield`)
    - `internal/left/right`: căn lề (`adjustfield`)
  - `width(int w)`: thay đổi độ rộng của trường
  - `precision(int p)`: thay đổi độ chính xác
  - Ví dụ:

```
cout.width(15);
cout.setf(ios::right, ios::adjustfield);
cout.setf(ios::scientific, ios::floatfield);
cout.precision(3);
cout << 34.5678;
```

- Dùng các manipulator:
  - Bao gồm: `internal`, `left`, `right`, `dec`, `hex`, `oct`, `fixed`, `scientific`, `setprecision(p)`, `setw(w)`, `setiosflag(flags)`, `endl`, `ends`, `flush`
  - `#include <iomanip>`
  - Ví dụ:

```
cout << setw(15) << right << scientific
     << setprecision(3) << 34.5678;
```

# Đọc/ghi file

- `#include <fstream>`
- Sử dụng `ifstream` (file chỉ đọc), `ofstream` (chỉ ghi), `fstream` (đọc/ghi)
- Đọc/ghi dữ liệu dùng các toán tử >> và << tương tự như với vào/ra chuẩn
- Mở file:
  - `ifstream f1("ten file", ios::in | ios::binary);`
  - `ofstream f2;`
    `f2.open("ten file", ios::out | ios::trunc);`
  - Các mode:

| | | | |
|---|---|---|---|
| `app` | Luôn nhảy con trỏ tới cuối file khi ghi | `trunc` | Xoá nội dung cũ khi mở |
| `ate` | Con trỏ tới cuối file | `binary` | File nhị phân |
| `in` | Cho phép đọc | `out` | Cho phép ghi |

- Đóng file:
  - `f.close();`
  - Có thể để đóng file tự động trong destructor khi các đối tượng bị huỷ
- Chú ý khi dùng fstream để dùng cả đọc và ghi: trước khi chuyển từ việc đọc sang ghi hoặc ngược lại, phải dùng hàm `seekg/seekp(...)`

# Đọc/ghi file dạng nhị phân

- Mở file: thêm cờ `ios::binary`
- Đọc dữ liệu:
  - `file.read(char* buffer, int size)`

    `file.gcount()  // số byte đã được đọc`
- Ghi dữ liệu:
  - `file.write(char* buffer, int size)`
- Kiểm tra lỗi đọc/ghi:
  - `file.read/write(...)`

    `if (!file) {...}`
  - `if (!file.read/write(...)) {...}`
- Di chuyển con trỏ file: C++ phân biệt con trỏ đọc và con trỏ ghi
  - Di chuyển con trỏ đọc file: `file.seekg(int pos, ios::beg/cur/end)`
  - Vị trí con trỏ đọc hiện tại: `file.tellg()`
  - Di chuyển con trỏ ghi file: `file.seekp(int pos, ios::beg/cur/end)`
  - Vị trí con trỏ ghi hiện tại: `file.tellp()`

# Đọc/ghi file: ví dụ copy file

```
bool copy_file(const char* src, const char* dst) {
    ifstream fs(src, ios::in | ios::binary);
    ofstream fd(dst, ios::out | ios::binary
                        | ios::trunc);
    if (!fs || !fd) return false;

    char buf[1024];
    while (fs) {
       fs.read(buf, sizeof(buf));
       fd.write(buf, fs.gcount());
    }
    return true;
}
```

# Vào/ra với chuỗi

- `#include <sstream>`
- Sử dụng `istringstream` (chỉ đọc), `ostringstream` (chỉ ghi), `stringstream` (đọc/ghi)
- Đọc từ chuỗi:
  - ```
    string s("10 3.56 y");
    istringstream str(s);
    int i;  double d;  char c;
    str >> i >> d >> c;
    ```
  - Dùng để trích dữ liệu từ chuỗi
- Ghi ra chuỗi:
  - ```
    ostringstream str;
    str << "i=" << i << ", d=" << d << ", c=" << c;
    string s = str.str();
    ```
  - Dùng để định dạng dữ liệu ra chuỗi

# Định nghĩa toán tử >> và <<

- Việc xuất/nhập dữ liệu dựa trên định nghĩa chồng các toán tử >> và <<

```
ostream& operator <<(ostream& os, char c);
ostream& operator <<(ostream& os, const char* s);
ostream& operator <<(ostream& os, double n);
...
istream& operator >>(istream& is, char& c);
istream& operator >>(istream& is, const char* s);
istream& operator >>(istream& is, double& n);
...
```

- Các toán tử này trả về chính đối tượng ostream/istream nhận ở tham số để có thể móc nối nhiều lần:

```
is >> a >> b >> c;
os << a << b << c;
```

- Cần định nghĩa các toán tử này cho các lớp mới định nghĩa:

```
ostream& operator <<(ostream& os, const Ellipse& e) {
   return os << e.rx << e.ry;
}
istream& operator >>(istream& is, Ellipse& e) {
   return is >> e.rx >> e.ry;
}
```

# Khái niệm về serialize (tuần tự hoá)

- Là việc chuyển đổi một đối tượng có cấu trúc dữ liệu bất kỳ thành một luồng thông tin có thứ tự để có thể ghi ra rồi đọc lại
- Ứng dụng trong việc truyền tin và lưu trữ dữ liệu
- Với STL, ta có thể định nghĩa các toán tử >> và << để thực hiện serialize
- Ví dụ:

```
istream& operator >>(istream& is, SinhVien& sv) {
   return is >> sv.ten >> sv.khoa >> sv.nam_sinh;
}
ostream& operator <<(ostream& os, SinhVien& sv) {
   return os << sv.ten << sv.khoa << sv.nam_sinh;
}
```

# Bài tập

1. Viết chương trình nhập mảng số nguyên có số phần tử bất kỳ từ bàn phím rồi in ra các số chẵn bằng cách duyệt mảng
2. Sửa bài trên thay duyệt mảng bằng dùng hàm `find_if(...)` để in ra các số chẵn
3. Sửa lại chương trình trên để dùng DSLK thay cho mảng
4. Cho hai mảng a1 và a2 đều có giá trị tăng dần, viết hàm trộn hai mảng này thành mảng a3 cũng có giá trị tăng dần
5. Đọc dữ liệu từ file và lưu dưới dạng danh sách các dòng, sau đó in ra các dòng có độ dài từ 10 đến 20 ký tự
6. Định nghĩa toán tử << và >> cho lớp Fraction và thử dùng nó để xuất/nhập dữ liệu với cin/cout, file, chuỗi
7. Định nghĩa các toán tử << và >> cho lớp Complex để đọc/ghi dữ liệu dưới dạng nhị phân