CÁC KÊNH XUẤT NHẬP

1. GIỚI THIỆU CHUNG

1.1 Khái niệm về kênh

Trong các chương trước, chúng ta thường sử dụng các chỉ thị viết ra thiết bị ra chuẩn như :

```
cout<<n;
```

Chỉ thị này gọi đến toán tử "<<" và cung cấp cho nó hai toán hạng, một tương ứng với "kênh xuất - output stream"(ở đây là **cout**), toán hạng thứ hai là biểu thức mà chúng ta muốn viết giá trị của nó (ở đây là `n`).

Tương tự, các chỉ thị đọc từ thiết bị vào chuẩn kiểu như:

```
cin >> x;
```

gọi tới toán tử ">>" và cung cấp cho nó hai toán hạng, một là "kênh nhập-input stream"(ở đây là **cin**), còn toán hạng thứ hai là một biến mà ta muốn nhập giá trị cho nó.

Một cách tổng quát, một kênh(stream) được hiểu như một kênh truyền:

(i) nhận thông tin, trong trường hợp ta nói đến dòng xuất

(ii) cung cấp thông tin, trong trường hợp ta nói đến dòng nhập.

Các toán tử "<<" và ">>" ở đây đóng vai trò chuyển giao thông tin, cùng với khuôn dạng của chúng.

Một kênh có thể được nối với một thiết bị ngoại vi hoặc một tập tin. Kênh **cout** được định nghĩa nối đến thiết bị ra chuẩn (tương đương stdout). Cũng vậy, kênh **cin** được định nghĩa trước để nối đến thiết bị vào chuẩn(stdin). Thông thường **cout** tương ứng với màn hình, còn **cin** thì đại diện cho bàn phím. Tuy nhiên trong trường hợp cần thiết thì có thể đổi hướng các vào ra chuẩn này đến một tập tin.

Ngoài các kênh chuẩn **cin** và **cout,** người sử dụng có thể định nghĩa cho mình các kênh xuất nhập khác để kết nối với các tập tin.

1.2 Thư viện các lớp vào ra

C++ cung cấp một thư viện các lớp phục vụ cho công việc vào ra. Lớp

streambuf là cơ sở cho tất cả các thao tác vào ra bằng toán tử; nó định nghĩa các đặc trưng cơ bản của các vùng đệm lưu trữ các ký tự để xuất hay nhập. Lớp ios là lớp dẫn xuất từ streambuf, ios định nghĩa các dạng cơ bản và khả năng kiểm tra lỗi dùng cho streambuf. ios là một lớp cơ sở ảo cho các lớp istream và ostream. Mỗi lớp này có định nghĩa chồng toán tử "<<" và ">>" cho các kiểu dữ liệu cơ sở khác nhau. Để sử dụng các khả năng này phải dùng chỉ thị #include đối với tập tin tiêu đề iostream.h. Cơ chế lớp của C++ cho phép tạo ra hệ thống giao tiếp có khả năng mở rộng và nhất quán. Trong chương 4 đã đưa ra hai định nghĩa chồng cho các toán tử vào/ra trong C++.

Phụ lục này tập trung trình bày các khả năng vào ra do C++ cung cấp, bao gồm các nội dung sau:

(iii) khả năng của ostream, istream,

(iv) kiểm soát lỗi vào ra

2. LỚP OSTREAM

2.1 Định nghĩa chồng toán tử << trong lớp ostream

Trong lớp ostream, toán tử "<<" được định nghĩa cho tất cả các kiểu dữ liệu cơ sở dưới dạng một hàm toán tử thành phần:

```
ostream &operator<<(expression)
```

trong đó expression có kiểu cơ sở bất kỳ. Vai trò của hàm toán tử này là chuyển giá trị của biểu thức tới kênh liên quan, đồng thời định dạng giá trị đó một cách thích hợp. Xét chỉ thị:

```
cout<<n;
```

Nếu n có giá trị 1234, toán tử "<<" sẽ chuyển đổi giá trị nhị phân của n sang hệ thập phân và chuyển đến **cout** các ký tự tương ứng với các chữ số của số thập phân nhận được (ở đây là 1, 2, 3, 4).

Ngoài ra, toán tử này trả về giá trị là tham chiếu đến kênh xuất đã gọi nó, sau khi thông tin được viết ra. Do vậy, cho phép viết liên tiếp nhiều giá trị lên cùng một kênh:

```
cout<<"Gia tri : "<<n<<"\n";
```

2.2 Hàm put

Hàm thành phần put trong lớp ostream dùng để đưa ra kênh xuất tham số ký tự. Chỉ thị

```
cout.put(c);
```

sẽ đưa ra kênh xuất **cout** ký tự `c`.

Giá trị trả về của `put` là tham chiếu đến kênh xuất đang sử dụng. Có thể ghi liên tiếp các ký tự trên cùng kênh xuất như sau:

```
cout.put(c1).put(c2).put(c3);
```

Chỉ trị trên tương đương với ba chỉ thị riêng biệt:

```
cout.put(c1);
cout.put(c2);
cout.put(c3);
```

2.3 Hàm write

Hàm thành phần `write` cho phép ghi ra kênh xuất một chuỗi các ký tự có chiều dài đã cho. Ví dụ, với:

```
char t[] = "hello";
```

chỉ thị

```
cout.write(t, 4);
```

sẽ gửi đến **cout** bốn ký tự đầu tiên của xâu `t` là `h e l l`.

Giống như `put`, hàm `write` trả về giá trị là tham chiếu đến chính kênh xuất vừa nhận thông tin. Tương tự, có thể gọi liên tiếp các hàm `write` như đối với hàm `put`:

```
//in ra ba ký tự đầu tiên của xâu t.
cout.write(t,1).write(t+1,1).write(t+2,1);
```

2.4 Khả năng định dạng

2.4.1 Chọn cơ số thể hiện

Khi viết một giá trị số nguyên, cơ số ngầm định để biểu diễn giá trị là hệ đếm thập phân. Tuy nhiên ta có thể lựa chọn các cơ số khác nhau để hiển thị giá trị: 10 (decimal), 16 (hexa decimal), 8 (octal). Chương trình `io1.cpp` sau đây đưa ra một ví dụ minh hoạ:

Ví dụ

```
/*io1.cpp*/
#include <iostream.h>
```

```
#include <conio.h>
void main() {
 clrscr();
 int n = 12000;
 cout<<"Ngam dinh "   <<n<<endl;
 cout<<"Duoi he 16 "<<hex<<n<<endl;
 cout<<"Duoi he 10 "<<dec<<n<<endl;
 cout<<"Duoi he 8 "<<oct<<n<<endl;
 cout<<"va ...   "   <<n<<endl;
 getch();
 }
```

```
Ngam dinh  12000
Duoi he 16 2ee0
Duoi he 10 12000
Duoi he 8  27340
va ...    27340
```

Các ký hiệu `hex, dec, oct` được gọi là toán tử định dạng. Đó chính là các toán tử đã được định nghĩa trước trong `ostream`. Các toán tử này chỉ có một toán hạng là đối tượng `ostream` và trả về chính đối tượng đó sau khi nó thực hiện một số tho tác nhất định. Trong ví dụ này, thao tác được thực hiện là thay đổi cơ số hiển thị giá trị và thông tin về cơ số sẽ được ghi lại trong `ostream` cho các lần thực hiện tiếp sau.

2.4.2 Đặt độ rộng

Lớp `ostream` cung cấp cho người sử dụng các phương thức hoặc các toán tử để kiểm soát cách thức máy tính định dạng xuất và nhập các giá trị. Để xác định độ rộng của trường để hiện thị thông tin ta sử dụng phương thức `width`. Xét các chỉ thị sau:

```
int x = 10;
cout.width(5);
cout<<x;
```

Giá trị của `x` sẽ được hiển thị sát lề phải trong trường với độ rộng 5 ký tự. Nếu kích thước của `x` lớn hơn độ rộng đã đặt thì giá trị đã định của độ rộng sẽ bị bỏ qua và toàn bộ giá trị của `x` sẽ được hiển thị. Giá trị ngầm định của độ rộng cho một

kênh xuất nhập là 0, nghĩa là dữ liệu được xuất ra theo kích thước thực tế mà không độn thêm ký tự gì. Sau mỗi lần xuất, độ rộng sẽ được đặt lại giá trị là 0. Đoạn chương trình sau:

```
int x = 1, y =2;
cout.width(5);
cout<<x<<' '<<y;
```

sẽ xuất ra giá trị của `x` trong một trường có 5 ký tự, sau đó là một dấu trắng và giá trị của `y` với kích thước thực tế của nó. `width` cũng không phải là phương thức duy nhất được dùng để thay đổi đặc tính của các kênh xuất/nhập. Xét các chỉ thị sau:

```
float pi=3.1415927;
int orig_prec = cout.precision(2);
cout<<pi;
cout.precision(orig_prec);
```

Trong đoạn chương trình trên, phương thức `precision` dùng để xác định lại số chữ số sẽ được in ra sau dấu chấm thập phân cho các giá trị thực. Phương thức `precision` có thể có tham số và sẽ trả về số chữ số thập phân thực có đứng sau dấu chấm. Giá trị ngầm định cho `precision` là 6.

Trong trường hợp giá trị đặt cho độ rộng lớn hơn chiều dài của giá trị được xuất ra, thì dùng các ký tự độn để lấp khoảng trống. Ký tự độn ngầm định là dấu cách. Tuy vậy có thể sử dụng phương thức `fill` để sử dụng một ký tự khác dấu trắng. Đoạn chương trình sau:

```
int x = 10;
cout.fill('0');
cout.width(5);
cout<<x;
```

cho kết quả như sau:

```
00010
```

Có thể thay thế phương thức `width` trong **cout**, **cin** bằng toán tử `setw` như sau:

thay vì sử dụng

```
cout.width(5);
cout<<x;
```

ta dùng

```
cout<<setw(5)<<x;
```

3. LỚP ISTREAM

3.1 Định nghĩa chồng toán tử ">>" trong lớp istream

Trong lớp `istream`, toán tử ">>" được định nghĩa chồng để có thể làm việc với tất cả các kiểu dữ liệu cơ sở (bao gồm cả **char** *) dưới dạng hàm thành phần:

```
istream & opertaor >>(&base_type)
```

Theo khai báo này toán tử ">>" có hai toán hạng, toán hạng đứng bên trái sẽ là đối tượng kiểu `istream`, đối tượng này sẽ là tham số ngầm định cho hàm toán tử. Toán hạng đứng bên phải ">>" là tham chiếu đến biến kiểu cơ sở sẽ nhập giá trị. Thực hiện ">>" sẽ cho kết quả là một tham chiếu đến đối tượng có kiểu `istream`. Thông thường, đó chính là đối tượng kênh nhập dữ liệu. Cũng như đối với **cout** và "<<", toán tử ">>" cũng cho phép nhập liên tiếp các biến khác nhau.

Các dấu phân cách bao gồm: `' ' '\t' '\v' '\n' '\r' '\f'` sẽ không được xem xét khi đọc; chẳng hạn, xét vòng lặp thực hiện chỉ thị (trong đó c có kiểu ký tự char):

```
cin >>c;
```

Với đầu vào có dạng

```
x i
n  c
h a o
```

thì chỉ có các ký tự `x, i, n, c, h, a, o` được đọc.

Để đọc được các ký tự trắng, phải sử dụng hàm thành phần `get` trong `istream`.

Mặt khác khi đọc một xâu ký tự không thể đọc các dấu trắng trong xâu . Chẳng hạn, với nội dung của dòng nhập là

"Xin chao"

thì chỉ lấy được phần đầu "Xin" trong xâu này để làm nội dung.

Để có thể đọc được các xâu có chứa dấu phân cách sử dụng hàm thành phần getline định nghĩa trong lớp istream.

3.2 Hàm thành phần get

Hàm thành phần

```
istream & get( char &);
```

cho phép đọc một ký tự từ kênh nhập và gán nó cho biến có kiểu ký tự (là tham số của hàm). Hàm này trả về giá trị là một tham chiếu đến kênh nhập, nên có thể gọi get liên tiếp để đọc nhiều ký tự.

Khác với toán tử ">>", hàm get có thể đọc tất cả các ký tự kể cả là các dấu phân cách. Bạn đọc có thể kiểm tra số ký tự đọc được nhờ sử dụng get đối với dòng nhập có nội dung:

```
x i
n  c


hao
```

Khi gặp EOF (hết dòng nhập) hàm get trả về giá trị 0. Xét đoạn chương trình sau:

```
char c;
...
while (cin.get(c))  //chép lại dòng nhập cin
 cout.put(c);     // lên dòng xuất cout
                          // công việc sẽ dừng khi eof vì khi đó (cin)=0
```

Còn một hàm thành phần get khác của lớp istream:

```
int get()
```

Khi gặp dấu kết thúc tập tin, hàm trả về giá trị EOF còn bình thường hàm đưa lại ký tự đọc được.

3.3 Các hàm thành phần `getline` và `gcount`

Hai hàm này sử dụng để đọc các xâu ký tự.

Khai báo của `getline` có dạng:

```
istream & getline(char * ch, int size, char delim='\n')
```

Hàm `getline` đọc các ký tự trên kênh nhập gọi nó và đặt vào vùng nhớ có địa chỉ xác định bởi `ch`. Hàm bị ngắt khi:

- ký tự phân cách `delim` xuất hiện trong dòng nhập

- hoặc đã đọc đủ `size-1` ký tự.

Trong cả hai trường hợp, hàm này bổ sung thêm một ký tự kết thúc xâu ngay sau các ký tự đọc được (xem lại hàm `gets()` trong `stdio.h`).

Ký tự phân cách `delim` có giá trị ngầm định là '`\n`' khi đọc các dòng văn bản.

Hàm `gcount` cho biết số ký tự được đọc trong chỉ thị gọi hàm `getline` gần nhất, ở đây không tính tới ký tự phân cách cũng như ký tự cuối xâu được thêm vào tự động.

Xem các chỉ thị sau:

```
const LG_LIG = 120; // chiều dài cực đại của một dòng
...
char ch[LG_LIG+1]; //khai báo 1 dòng
int lg;
...
while(cin.getline(ch,LG_LIG)) {
  lg = cin.gcount();
  //xử lý một dòng có lg ký tự
  }
```

3.4 Hàm thành phần `read`

Hàm `read` cho phép đọc từ kênh nhập một dãy ký tự có chiều dài xác định. Chẳng hạn, với:

```
char t[10];
```

chỉ thị

```
cin.read(t,5);
```

sẽ đọc từ thiết bị vào 5 ký tự và đưa vào 5 ký tự đầu tiên của mảng các ký tự `t`.

Hàm `read` không phân biệt dấu trắng với các ký tự khác trên kênh nhập.

3.5 Một số hàm khác

Hàm `putback(char c)` cho phép trả lại kênh nhập một ký tự c (tham số của hàm).

Hàm `peek()` đưa ra ký tự tiếp theo trong dòng nhập nhưng không lấy ký tự đó ra khỏi dòng nhập.

4. TRẠNG THÁI LỖI CỦA KÊNH NHẬP

Mỗi kênh nhập hay xuất đều có một số cờ xác định trạng thái lỗi của kênh hiện tại. Trong mục này trước hết ta sẽ xem xét ý nghĩa của các cờ này, sau đó sẽ tìm hiểu cách để lấy giá trị của chúng và thay đổi các giá trị của các cờ theo mục đích của chúng ta. Cuối cùng ta xem xét định nghĩa chồng các phép toán () và ! nhằm đơn giản hoá cách sử dụng một kênh dữ liệu.

4.1 Các cờ lỗi

Các cờ lỗi được định nghĩa như là các hằng trong lớp `ios` dẫn xuất từ `ostream` và `istream`. Đó là:

eofbit	Kết thúc tập tin; cờ này được kích hoạt nếu gặp dấu kết thúc tập tin. Nói cách khác khi kênh nhập không còn ký tự để đọc tiếp nữa.
failbit	Bit này được bật khi thao tác vào ra tiếp theo không thể tiến hành được.
badbit	Bit này được bật khi kênh ở trạng thái không thể khôi phục được.

`failbit` và `badbit` chỉ khác nhau đối với các kênh nhập. Khi `failbit` được kích hoạt, các thông tin trước đó trong kênh nhập không bị mất; trong khi đó điều này không còn đúng đối với `badbit` .

Ngoài ra, còn có cờ `goodbit` tương ứng với trạng thái không có lỗi.

Có thể nói rằng một thao tác vào ra thành công khi `goodbit` hay `eofbit` được bật. Tương tự, thao tác vào ra tiếp theo chỉ được tiến hành nếu `goodbit` được bật.

Khi một dòng ở trạng thái lỗi, mọi thao tác tiếp theo phải chờ cho đến khi:

- trạng thái lỗi được sửa chữa,

- các cờ lỗi được tắt.

Ta sẽ xem xét các hàm thực hiện các công việc này trong các mục dưới đây.

4.2 Các thao tác trên các bit lỗi

Có hai loại hàm thành phần thực hiện các thao tác này:

(v) Các hàm thành phần cho phép giá trị các cờ lỗi,

(vi) Các hàm thành phần cho phép bật tắt các cờ lỗi đó.

4.2.1 Đọc giá trị

Trong lớp `ios` có định nghĩa năm hàm thành phần sau đây:

eof()	trả về 1 nếu gặp dấu kết thúc file, có nghĩa là `eofbit` được kích hoạt.
bad()	trả về 1 nếu `badbit` được bật.
fail()	trả về 1 nếu `failbit` được bật.
good()	trả về 1 nếu ba hàm trên cho giá trị 0
rdstate()	trả về một số nguyên tương ứng với tất cả các cờ lỗi.

4.2.2 Thay đổi trạng thái lỗi

Trong `istream/ostream` có hàm thành phần

```
void clear(int i = 0)
```

để bật các bit lỗi tương ứng với giá trị được sử dụng làm tham số. Thông thường, ta xác định giá trị đó dựa trên các hằng số của các cờ lỗi. Chẳng hạn, nếu `fl` biểu thị một kênh, chỉ thị:

```
fl.clear(ios::badbit);
```

sẽ bật cờ lỗi `badbit` và tắt tất cả các cờ còn lại.

Nếu ta muốn bật cờ này đồng thời không muốn thay đổi giá trị các cờ khác, sử dụng chỉ thị sau:

```
fl.clear(ios::badbit|fl.rdstate());
```

4.3 Định nghĩa các toán tử () và !

Có thể kiểm tra một kênh bằng cách xem nó như một giá trị logic. Điều này được thực hiện nhờ việc định nghĩa chồng trong lớp `ios` các toán tử () và !.

Chi tiết hơn, toán tử () được định nghĩa chồng dưới dạng (trong đó `fl` biểu thị một dòng):

```
(fl)
```

- trả về một giá trị khác 0 nếu các cờ lỗi được tắt, có nghĩa là hàm `good()` có giá trị bằng 1.
- trả về giá trị 0 trong trường hợp ngược lại, có nghĩa là khi `good()` có giá trị 0.

Như vậy:

```
if(fl) ...
```

có thể thay thế cho (hoặc được thay thế bởi)

```
if(fl.good()) ...
```

Cũng vậy, toán tử ! được định nghĩa như là

```
!fl
```

- trả về giá trị không nếu có ít nhất một cờ lỗi được bật lên
- trả về giá trị khác không trong trường hợp ngược lại.

Như vậy:

```
if ( !fl) ...
```

có thể thay thế (hoặc được thay thế bởi)

```
if (!flot.good()) ...
```

5. QUẢN LÝ ĐỊNH DẠNG

Các kênh xuất/nhập dùng giá trị cờ để điều khiển dạng nhập và xuất. Một ưu điểm của phương pháp quản lý định dạng sử dụng trong C++ là cho phép người lập trình bỏ qua tất cả các khía cạnh định dạng trong các chỉ thị đưa ra, nếu sử dụng các cờ ngầm định. Bên cạnh đó, khi có nhu cầu, người lập trình có thể đưa ra các định dạng thích hợp (một lần cho tất cả các chỉ thị vào/ra) với các loại dữ liệu.

5.1 Trạng thái định dạng của một dòng

Trạng thái định dạng của một dòng chứa:

- một từ trạng thái, trong đó mỗi bit có một ý nghĩa xác định
- các giá trị số mô tả giá trị của các hằng sau:

Độ rộng	Số ký tự để đưa thông tin ra. Giá trị này là tham số của `setw`, một toán tử định nghĩa trong `ostream`/`istream`. Khi giá trị này quá nhỏ, nó sẽ không có tác dụng nữa, các dòng sẽ hiển thị thông tin theo độ rộng bằng kích thước mà dữ liệu có.
Độ chính xác	Số chữ số được hiển thị sau dấu chấm thập phân trong dạng dấu chấm cố định và cũng là số ký tự có ý nghĩa trong ký pháp khoa học.
Ký tự thay thế	Nghĩa là các ký tự được sử dụng để điền thêm vào phần còn trống khi giá trị đưa ra không đủ để điền hết độ rộng. Ký tự thay thế ngầm định là dấu cách.

5.2 Từ trạng thái định dạng

Từ trạng thái định dạng được mô tả như một số nguyên trong đó mỗi bit (cờ) tương ứng với một hằng số định nghĩa trong lớp `ios`. Mỗi cờ định dạng được bật khi bit tương ứng có giá trị 1, trái lại ta nói cờ bị tắt. Giá trị của các cờ có thể sử dụng để:

- nhận diện bit tương ứng trong từ định dạng

- để tạo nên một từ trạng thái.

Các trường bit (ít nhất 3) được thay đổi giá trị mà không cần cung cấp tham số cho các hàm thành phần, là do chúng được định nghĩa ngay bên trong từ trạng thái, là đối tượng gọi hàm thành phần có tác dụng thay đổi nội dung các cờ.

Sau đây là danh sách các cờ kèm theo tên của trường bit tương ứng.

Tên trường	**Tên bit**	**Ý nghĩa**
	`ios::skipws`	Bỏ qua các dấu phân cách (khi nhập)
`ios::adjustfield`	`ios::left`	Căn lề bên trái (xuất)
	`ios::right`	Căn lề bên phải (xuất)
	`ios::internal`	Các ký tự độn được điền giữa dấu và giá trị
`ios::basefield`	`ios::dec`	Cơ số hiển thị là cơ số 10
	`ios::hex`	Cơ số hiển thị là cơ số 16
	`ios::oct`	Cơ số hiển thị là cơ số 8
	`ios::showbase`	

ios::showpoint	Hiển thị các chữ số 0 sau các số thập phân ngay cả khi chúng không có. Ngầm định cờ này không được bật.
ios::uppercase	Tất cả các chữ số hiển thị sẽ được chuyển đổi sang chữ in.
ios::showpos	Dấu + sẽ được xuất ra trước bất kỳ số nguyên nào. Ngầm định cờ này không được bật.
ios::scientific	Khi được bật, các giá trị dấu phẩy động sẽ được xuất ra theo dạng khoa học. Sẽ chỉ có một con số đứng trước dấu chấm thập phân và các con số thập phân có nghĩa sẽ đi sau nó, sau đó là chữ "e" ở dạng chữ hoa hày thường (tuỳ thuộc cờ uppercase), theo sau là giá trị số mũ.
ios::fixed	Khi được bật, giá trị được xuất ra theo dạng số thập phân, có các chữ số thập phân theo sau dấu chấm thập phân. Nếu không bật cả hai cờ thì dạng biểu diễn khoa học sẽ dùng khi số mũ nhỏ hơn -4 hoặc lớn hơn giá trị được mô tả bởi precision
ios::unibuf	Khi được bât, kênh xuất nhập được thiết lập lại sau mỗi lần xuất ra. Cờ này không bật theo ngầm định.
ios::stdio	Cờ này được sử dụng để dọn dẹp các thiết bị xuất stdout và stderr.

5.3 Thao tác trên trạng thái định dạng

Để tác động lên các trạng thái định dạng, có thể sử dụng các toán tử thao tác định dạng hoặc sử dụng các hàm thành phần của các lớp istream và ostream.

Tuỳ theo từng trường hợp, các thao tác này có thể tác động lên toàn bộ từ trạng thái hay chỉ các giá trị: độ rộng, độ chính xác, ký tự độn. Bên cạnh đó còn có các hàm thành phần cho phép chúng ta lưu giữ giá trị các trạng thái định dạng để khôi

phục lại về sau.

5.3.1 Các toán tử thao tác định dạng không tham số (TTĐDKTS)

Đây là các toán tử định dạng được sử dụng ở dạng sau (trong đó `fl` đóng vai trò một dòng nhập/ xuất, `manipulator` là toán tử định dạng):

```
fl<<manipulator
```

hay

```
fl>>manipulator
```

Kết quả thực hiện cho ta tham chiếu đến kênh hiện tại, do vậy đó cho phép xử lý chúng như cách thức chuyển thông tin. Đặc biệt nó còn cho phép áp dụng nhiều lần liên tiếp các toán tử "<<" và ">>".

Sau đây là danh sách các toán tử định dạng không tham số:

TTĐDKTS	Sử dụng trong các kênh	Hoạt động
`dec`	vào/ra	Kích hoạt cờ cơ số biểu diễn hệ 10
`hex`	vào/ra	Kích hoạt cờ cơ số biểu diễn hệ 16
`oct`	vào/ra	Kích hoạt cờ cơ số biểu diễn hệ 8
`ws`	vào	Kích hoạt cờ `skipws`
`endl`	ra	Thêm dấu xuống dòng
`ends`	ra	Thêm ký tự kết thúc xâu
`flush`	ra	Làm rỗng bộ đệm

5.3.2 Các toán tử định dạng có tham số(TTĐDCTS)

Các toán tử này được khai báo trong các lớp `ostream`, `istream` dưới dạng hàm thành phần:

```
istream &manipulator(argument)
```

hoặc

```
ostream &manipulator(argument)
```

Các toán tử này được sử dụng giống như các toán tử định dạng không có tham số. Tuy nhiên, muốn sử dụng chúng phải tham chiếu tập tin tiêu đề `iomaip.h` bằng chỉ thị:

```
#include <iomanip.h>
```

Sau đây là danh sách các toán tử định dạng có tham số:

TTĐDCTS	Sử dụng cho các dòng	Vai trò
`setbase(int)`	vào/ra	Định nghĩa cơ số hiển thị
`resetiosflags(long)`	vào/ra	Đặt lại 0 tất cả các bit có mặt trong tham số
`setiosflags(long)`	vào/ra	Kích hoạt các bit có trong tham số
`setfill(int)`	vào/ra	định nghĩa lại ký tự độn
`setprecision(int)`	vào/ra	Định nghĩa độ chính xác cho các số thực
`setw(int)`	vào/ra	Định nghĩa độ rộng

5.3.3 Các hàm thành phần

Trong hai lớp `istream` và `ostream` có bốn hàm thành phần: `setf, fill, precision,` và `width` được mô tả như sau:

Hàm `setf`

Hàm này cho phép thay đổi từ trạng thái định dạng. Hàm này có hai phiên bản khác nhau:

```
long setf(long)
```

lời gọi tới phiên bản này kích hoạt các cờ được mô tả trong tham số. Giá trị trả về của hàm là trạng thái cũ của từ trạng thái định dạng. Lưu ý rằng hàm này không tác động đến các cờ không được mô tả. Như vậy, với `fl` biểu thị một kênh, chỉ thị:

```
fl.setf(ios::oct);
```

sẽ kích hoạt cờ `oct`. Tuy nhiên, rất có thể các cờ khác như `dec` hay `hex` vẫn còn tác dụng. Dạng thứ hai của hàm `setf` hay được sử dụng trong thực tế là:

```
long setf(long, long)
```

Lời gọi tới phiên bản này kích hoạt các cờ mô tả trong tham số thứ nhất ở trong tham số thứ hai. Chẳng hạn, nếu `fl` là một kênh, chỉ thị sau:

```
fl.setf(ios::oct,ios::basedfield);
```

sẽ kích hoạt cờ `ios::oct` và tắt các cờ khác trong `ios::basefield`.

Giá trị trả về của lời gọi này là giá trị cũ của tham số thứ hai.

Hàm `fill`

Hàm này cho phép xác định và xác lập lại ký tự độn. Cũng có hai phiên bản khác nhau cho hàm này:

```
char fill()
```

Phiên bản này trả về ký tự độn hiện đang được sử dụng, trong khi đó

```
char fill(char)
```

được sử dụng để thay đổi ký tự độn.

Hàm `precision`

Hàm này cho phép xác định hoặc xác lập lại độ chính xác biểu diễn số thực. Hai phiên bản khác nhau cho hàm là:

```
int precision()
```

sẽ trả về giá trị mô tả độ chính xác hiện thời, còn

```
int precision(int)
```

đặt lại độ chính xác mới, đồng thời trả về giá trị cũ.

Hàm `width`

Hàm này cho phép xác định hay xác lập lại độ rộng của trường hiển thị thông tin. Cũng có hai phiên bản khác nhau:

```
int width()
```

sẽ trả về độ rộng đang được sử dụng hiện tại, còn

```
int width(int)
```

sẽ trả về độ rộng hiện thời đồng thời xác lập độ rộng mới là tham số được mô tả trong lời gọi hàm.

6. LIÊN KẾT KÊNH XUẤT/NHẬP VỚI MỘT TẬP TIN

Mục này trình bày cách để chuyển hướng vào ra tới một tập tin, đồng thời cũng giới thiệu các khả năng truy nhập trực tiếp vào các tập tin.

6.1 Liên kết xuất với một tập tin

Để liên kết một kênh xuất với một tập tin, ta chỉ cần tạo một đối tượng kiểu lớp `ofstream`, một lớp kế thừa từ `ofstream`. Việc sử dụng lớp này cần tới tập tin tiêu đề `fstream.h`.

Hàm thiết lập của lớp `ofstream` có hai tham số:

- tên của tập tin liên quan(dưới dạng một xâu ký tự)

- chế độ mở tập tin được xác định bởi một số nguyên.

Lớp ios có định nghĩa một số giá trị mô tả các chế độ mở tập tin khác nhau. Chỉ thị sau đây là một ví dụ minh hoạ:

```
ofstream output("abc.txt",ios::out);
```

Khi đó, đối tượng output sẽ được liên kết với tập tin tên là abc.txt, tập tin này được mở ở chế độ ghi. Sau khi đã tạo được một đối tượng ofstream, việc ghi ra tập tin được thực hiện giống như kết xuất ra một kênh xuất, chẳng hạn:

```
output<<12<<"abc"<<endl;
```

Có thể kiểm tra trạng thái lỗi của dòng xuất tương ứng với tập tin giống như cách ta đã dùng đối với các kênh xuất chuẩn:

```
if(output) ...
```

Chương trình ví dụ sau mô tả cách thức ghi một số số nguyên vào một tập tin.

```
/*io2.cpp*/
#include <stdlib.h>
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <conio.h>
const int LGMAX = 20;
void main() {
 clrscr();
 char filename[LGMAX+1];
 int n;
 cout<<"Ten tap tin : ";
 cin>>setw(LGMAX)>>filename;
 ofstream output(filename,ios::out);
 if (!output) {
   cout<<"Khong the tao duoc tap tin\n";
   exit(1);
   }
 do {
  cin >>n;
```

```
  if (n>0) output<<n<<' ';
  }while(n>0 && (output));
 output<<endl;
 output.close();
}
```

6.2 Liên kết kênh nhập với một tập tin

Một đối tượng của lớp `ifstream` sẽ được sử dụng để liên kết với một tập tin chứa thông tin cần nhập. Giống như `ofstream`, lớp `ifstream` cũng được định nghĩa trong tập tiêu đề `fstream.h`. Lớp `ifstream` có hàm thiết lập với hai tham số giống như `ofstream`. Chỉ thị sau đây sẽ liên kết một đối tượng `ifstream` với tập tin `abc.txt`:

```
ifstream input("abc.txt",ios::in);
```

Việc sử dụng `input` để đọc nội dung `abc.txt` giống hệt như việc sử dụng **cin** để đọc dữ liệu từ bàn phím. Ta xét ví dụ sau:

```
/*io3.cpp*/
#include <stdlib.h>
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <conio.h>
const int LGMAX = 20;
void main() {
 clrscr();
 char filename[LGMAX+1];
 int n;
 cout<<"Ten tap tin : ";
 cin>>setw(LGMAX)>>filename;
 ifstream input(filename,ios::in);
 if (!input) {
   cout<<"Khong the mo duoc tap tin\n";
   exit(1);
```

```
   }
 while(input) {
   input>>n;
   cout<<n<<endl;
   }
 input.close();
}
```

Nhận xét

Lớp `fstream` (thừa kế từ hai lớp `ifstream` và `ofstream`) dùng để định nghĩa các kênh dữ liệu thực hiện đồng thời cả hai chức năng nhập và xuất trên một tập tin. Việc khai báo một đối tượng kiểu `fstream` cũng giống như khai báo đối tượng `ofstream` và `ifstream`. Chỉ thị:

```
fstream file("abc.txt",ios::in|ios::out);
```

sẽ gắn đối tượng file với tập tin `abc.txt`, được mở để đọc và ghi đồng thời.

6.3 Các khả năng truy nhập trực tiếp

Việc truy nhập (đọc/ghi) đến tập tin dựa trên một phần tử là con trỏ tập tin. Tại mỗi thời điểm, con trỏ tập tin xác định một vị trí tại đó thực hiện thao tác truy nhập. Có thể xem con trỏ này như cách đếm số phim trong máy ảnh. Sau mỗi một thao tác truy nhập, con trỏ tập tin tự động chuyển sang vị trí tiếp theo giống như việc lên phim mỗi khi bấm máy ảnh. Ta gọi cách truy nhập tập tin kiểu này là truy nhập tuần tự. Các chương trình `io2.cpp`, `io3.cpp` sử dụng cách truy nhập này để đọc và ghi thông tin trên các tập tin. Nhược điểm của cách truy nhập tuần tự là phải đi từ đầu tập tin qua các tất cả các phần tử có trong tập tin để đi đến được phần tử cần thiết, do vậy tốn không ít thời gian. Cách truy nhập trực tiếp sẽ cho phép đến thẳng tới phần tử chúng ta cần nhờ sử dụng một số hàm thành phần thích hợp trong các lớp `ifstream` và `ofstream`.

Trong lớp `ifstream` có hàm `seekg` và trong lớp `ofstream` có hàm `seekp` được dùng để di chuyển con trỏ tập tin. Mỗi hàm thành phần đó có hai tham số:

- Tham số thứ nhất là số nguyên mô tả dịch chuyển (tính theo byte) con trỏ bao nhiêu vị trí so với vị trí gốc, được mô tả bởi tham số thứ hai (xem hai hàm `fseek` trong `stdio.h`).

- Tham số thứ hai lấy một trong ba giá trị sau:

`ios::beg`	vị trí gốc là đầu tập tin
`ios::cur`	vị trí gốc là vị trí hiện thời của con trỏ tập tin

`ios::end`	vị trí gốc là cuối tập tin.

Hai hàm `tellg` (đối với `ifstream`) và `tellp` (đối với `ofstream`) dùng để xác định vị trí hiện thời của các con trỏ tập tin.

Chương trình sau đây minh hoạ khả năng truy nhập tập tin trực tiếp.

```
/*io4.cpp*/
#include <stdlib.h>
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <conio.h>
const int LGMAX = 20;
void main() {
 clrscr();
 char filename[LGMAX+1];
 int n,num;
 cout<<"Ten tap tin : ";
 cin>>setw(LGMAX)>>filename;
 ifstream input(filename,ios::in);
 if(!input) {
  cout<<"Khong mo duoc tap tin";
  exit(1);
  }
 do {
  cout<<"So thu tu cua so nguyen se tim : ";
  cin>>num;
  if (num >0) {
   input.seekg(sizeof(int)*(num-1),ios::beg);
   input>>n;
   if (input) cout<<"--Gia tri : "<<n<<endl;
   else {
        cout<<"--Loi\n";
        input.clear();
```

```
        }
    }
  } while(num);
  input.close();
}
```

```
Ten tap tin : abc.txt
So thu tu cua so nguyen se tim : 3
--Gia tri : 3
So thu tu cua so nguyen se tim : 2
--Gia tri : 22
So thu tu cua so nguyen se tim : 1
--Gia tri : 1
So thu tu cua so nguyen se tim : 3
--Gia tri : 3
So thu tu cua so nguyen se tim : 4
--Gia tri : 3
So thu tu cua so nguyen se tim : 5
--Gia tri : 4
So thu tu cua so nguyen se tim : 6
--Gia tri : 3
So thu tu cua so nguyen se tim : 7
--Gia tri : 2
So thu tu cua so nguyen se tim : 6
--Gia tri : 3
So thu tu cua so nguyen se tim : 100
--Loi
So thu tu cua so nguyen se tim : 0
```

6.4 Các chế độ mở tập tin khác nhau

Chế độ	Mô tả hành động tương ứng
`ios::in`	Mở một tập tin để đọc(bắt buộc đối với `ifstream`)
`ios::out`	Mở một tập tin để ghi(bắt buộc đối với `ofstream`)
`ios::app`	Mở một tập tin để gắn thêm các thông tin vào cuối.
`ios::ate`	Đặt con trỏ tập tin vào cuối tập tin
`ios::trunc`	Nếu tập tin đã có, nội dung của nó sẽ bị mất.
`ios::nocreate`	Tập tin bắt buộc phải tồn tại.
`ios::noreplace`	Tập tin chưa tồn tại
`ios::binary`	Tập tin được mở ở chế độ nhị phân [1]
`ios::text`	Tập tin được mở ở chế độ văn bản.

Để thực hiện được nhiều hành động trên cùng một tập tin phải tổ hợp các bit mô tả chế độ bằng cách sử dụng toán tử | (cộng bit).

Chẳng hạn:

```
fstream f("abc.txt",ios::in|ios::out);
```

[1] Trong các môi trường dos và windows người ta phân biệt các tập tin văn bản và tập tin nhị phân. Khi mở một tập tin phải xác định ngay là liệu chúng ta sẽ làm việc với các tập tin loại nào. Sự phân biệt này về thực chất liên quan đến việc xử lý ký tự cuối dòng.