Stream Fail State

• possible reasons for entering fail state include:
  • invalid input data (often the wrong type)
  • i.e., expect an integer get a ‘z’

• when a stream enters the fail state, further I/O operations using that stream have no effect at all.
  – The computer does not automatically halt the program or give any error message
I/O states

- consists of 3 bits

- `eof()` – nonzero value if eofbit is set
- `bad()` – nonzero if badbit is set
- `fail()` – nonzero if badbit or failbit or eofbit is set

- `good()` – value of 0 (no bits set)
What they mean

**cin.fail()**: true if operation failed  
**cin.bad()**: true if stream is corrupt

**cin.good()**: true if no io_flag bits were set

i.e.

```cpp
cin.good() = !cin.eof() &&  
             !cin.fail() &&  
             !cin.bad()
```

```cpp
cin.clear();  //use this to clear all the bad bits!
```
Wrong Data Type – Example

double altitude;
char junk;

cout << “Enter the altitude: ”;
cin >> altitude;
Wrong Data Type – Example

double altitude;
char junk;

cout << “Enter the altitude: ”;
cin >> altitude;   // what happens if user enters ‘A’?
Wrong Data Type – Example

double altitude;
char junk;

cout << "Enter the altitude: ";
cin >> altitude;   // what happens if user enters 'A'?

if ( cin.fail() ) {

}
Wrong Data Type – Example

double altitude;
char junk;

cout << "Enter the altitude: ";
cin >> altitude;  // what happens if user enters 'A'?

if ( cin.fail() ) {
    cin.clear();
    cin >> junk;
    cout << "You did not enter a number" << endl;
    cout << "Enter the altitude: ";
    cin >> altitude;
}

Wrong Data Type – Example

double altitude;
char junk;

cout << "Enter the altitude: ";
cin >> altitude;   // what happens if user enters ‘A’?

if ( cin.fail() ) {
    cin.clear();

    cin >> altitude;
}

Wrong Data Type – Example

double altitude;
char junk;

cout << “Enter the altitude: ”;
cin >> altitude;  // what happens if user enters ‘A’?

if ( cin.fail() ) {
    cin.clear();
    cin >> junk;

    cin >> altitude;
}


Wrong Data Type – Example

double altitude;
char junk;

cout << “Enter the altitude: ”;
cin >> altitude;    // what happens if user enters ‘A’?

if ( cin.fail() ) {
    cin.clear();
    cin >> junk;
    cout << “You did not enter a number” << endl;
    cout << “Enter the altitude: ”;
    cin >> altitude;
}
Wrong Data Type – Example

double altitude;
char junk;

cout << “Enter the altitude: ”;
cin >> altitude;   // what happens if user enters ‘A’?

if ( cin.fail() ) {
    cin.clear();
    cin >> junk;
    cout << “You did not enter a number” << endl;
    cout << “Enter the altitude: ”;
    cin >> altitude;
}
Wrong Data Type – Example

double altitude;
char junk;

cout << “Enter the altitude: ”;
cin >> altitude;  // what happens if user enters ‘A’?

while ( cin.fail() ) {
cin.clear();
cin >> junk;
cout << “You did not enter a number” << endl;
cout << “Enter the altitude: ”;
cin >> altitude;
}

Wrong Data Type – Example

double altitude;
char junk;

cout << "Enter the altitude: ";
cin >> altitude;    // what happens if user enters ‘A’?

while ( cin.fail() ) {
    cin.clear();
    cin >> junk;
    cout << "You did not enter a number" << endl;
    cout << "Enter the altitude: ";
    cin >> altitude;
}
Formatting Output

- setw
- setprecision
- other formatting
Formatting Integer Output

```cpp
int numStudents = 36123;
cout << "UM has" << numStudents
  << " students.";

prints

UM has36123students.
```

- **default field width** is minimum required
- **field width**: number of spaces for value
- **default**: what happens when not explicitly specified
Explicit Field Width (setw)

```
int numStudents = 36123;
cout << "UM has" << setw(6) << numStudents << "students."
```

prints

UM has 36123students.

- Prints in field width 6, right-justified
- setw manipulator sets width for next item
- Output right-justified within field width
Wider Example

```cpp
int numStudents = 36123;
cout << "UM has" << setw(10) << numStudents << "students."
```

prints

```
UM has   36123students.
```

- Prints in field width 10, right-justified
Manipulating Justification

```cpp
int numStudents = 36123;
cout << left;  // flip to left justification
cout << "UM has" << setw(10) << numStudents << "students.";
```

prints

```
UM has36123     students.
```

- Prints in field width 10, left-justified
- Specified explicit justification policy
Default revisited

```cpp
int numStudents = 36123;
cout << "UM has" << numStudents << "students.";
```

prints

`UM has 36123 students.`
Default revisited

```cpp
int numStudents = 36123;
cout << "UM has " << numStudents
    << " students."
;
prints

UM has 36123 students.
```
Default revisited

```cpp
int numStudents = 36123;
cout << "UM has " << numStudents
    << " students.";
```

prints

UM has 36123 students.

- Typically the right thing for any #digits
- setw needed for lining up output (e.g., tables)
**Floating point output**

```c
float cost = 5.50;
cout << "Cost is $" << cost << " today."
```

prints

Cost is $5.50 today.

- default
  - large values printed in scientific notation
  - if number is whole, no decimal point
  - numbers of digits not under your control
Floating Point Manipulators

- **fixed**
  - decimal rather than scientific notation
  - `cout << fixed;`
- **showpoint**
  - show decimal point even if whole
  - `cout << showpoint;`
- **setprecision(n)**
  - show n decimal places
  - `cout << setprecision(2) << pi;`
- All of these stay in effect until changed (unlike `setw`)
- `setw only changes next value printed.`
Floating point output

```cpp
float cost = 5.50;
cout << "Cost is "$ << setw(5)
    << setprecision(2)
    << cost << "today."
    << "today."
```

prints

```
Cost is $ 5.50
today.
```

- if no field width is specified, minimum is used, just as for integers
Formatting char output

char ch = 'Q';
cout << '*' << ch << '*';

prints

*Q*

- default field width is 1
- setw has no effect
cout << 'H' << setw(10) << "ello";

prints

H      ello

• default field width is length of string
• can use setw
Required Libraries?

- For `endl`, `fixed`, `and` `showpoint`
  - `#include <iostream>`

- For `setw` and `setprecision`
  - `#include <iomanip>`
Useful output spacer

```cpp
string BLANK = " ";
cout << setw(10) << BLANK;
```

prints

10 Blanks
Useful output spacer

```
cont string BLANK = " ";
cout << setw(10) << BLANK;

prints

10 Blanks

cont char BLANK = ' ';
cout << setw(10) << BLANK;

prints

???
```
Useful output spacer

```cpp
const string BLANK = " ";
const char BLANK = ' ';
cout << setw(10) << BLANK;
```

prints

10 Blanks

```cpp
const char BLANK = ' ';
cout << setw(10) << BLANK;
```

prints

1 Blank
How would we generate this multiplication table?

<table>
<thead>
<tr>
<th>*</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>...</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>...</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>...</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
<td>...</td>
<td>27</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>18</td>
<td>27</td>
<td>36</td>
<td>45</td>
<td>54</td>
<td>...</td>
<td>81</td>
</tr>
</tbody>
</table>
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    int i, j;

    cout << " * | 1 2 3 4 5 6 7 8 9" << endl;
    cout << "------------------------------" << endl;
    cout << "    |" << endl;
    for (i = 1; i < 10; i++) {
        cout << setw(4) << i << " |";
        for (j = 1; j < 10; j++){
            cout << setw(4) << i * j;
        }
        cout << endl;
    }
    return 0;
}