USTC

Chapter 23
Text Processing

王子磊 (Zilei Wang)
Email: zlwang@ustc.edu.cn
http://vim.ustc.edu.cn

Overview

- 应用领域
- 字符串
- I/O
- Maps
- 规则表达式

A — Table 1: Special Characters

B / Character Name	Symbol	Shortcut
C — Cedilla	ç	Alt + 0231
Cicumflex	ˆ	Alt + 0136
Dagger	†	Alt + 0134
Ellipsis	…	Alt + 0133
oe ligature	œ	Alt + 0156
em dash	—	Alt + 0151 Ctrl+q, Shift+q

A. Table title B. Heading row C. Body rows

现在，你已经知道了基础知识

- 确实如此！恭喜你！

- 不要继续关注编程语言特性了！
  - 开始使用吧！
- 程序和应用是什么样的？通过编程能够做好什么？
  - 文本处理
  - 数值处理
  - 嵌入式系统编程
  - 银行
  - 医学应用
  - 科学可视化
  - 卡通制作
  - 路径规划
  - 物理设计

文本处理

- “我们表达的都是文本”
  - 往往都是的
- 书籍、论文
- 事务日志 (email, phone, bank, sales, …)
- Web 网页 (甚至是布局指令)
- 图表 (数字)
- 邮件
- 程序
- 测量
- 历史数据
- 医疗记录
- …

Amendment I

Congress shall make no law respecting an establishment of religion, or prohibiting the free exercise thereof; or abridging the freedom of speech, or of the press; or the right of the people peaceably to assemble, and to petition the government for a redress of grievances.

字符串概览

- 字符串
  - **std::string**
    - **<string>**
    - **s.size()**
    - **s1==s2**
  - C 风格字符串 (0结尾的字符数组)
    - **<cstring> 或 <string.h>**
    - **strlen(s)**
    - **strcmp(s1,s2)==0**
  - **std::basic_string<Ch>**, e.g. unicode 字符串
    - **typedef std::basic_string<char> string;**
  - 其它私有字符串类

字符串转换

- 简单的 to_string

```
template<class T> string to_string(const T& t)
{
  ostringstream os;
  os << t;
  return os.str();
}
```

- 例如：

```
string s1 = to_string(12.333);
string s2 = to_string(1+5*6-99/7);
```

字符串转换

- 简单地从字符串提取

```
template<class T> T from_string(const string& s)
{
    istringstream is(s);
    T t;
    if (!(is >> t)) throw bad_from_string();
    return t;
}
```

比 to_string 处理更复杂

- 例如：

```
double d = from_string<double>("12.333");
Matrix<int,2> m = from_string< Matrix<int,2> >("{ {1,2}, {3,4} }");
```

一般化的字符串转换

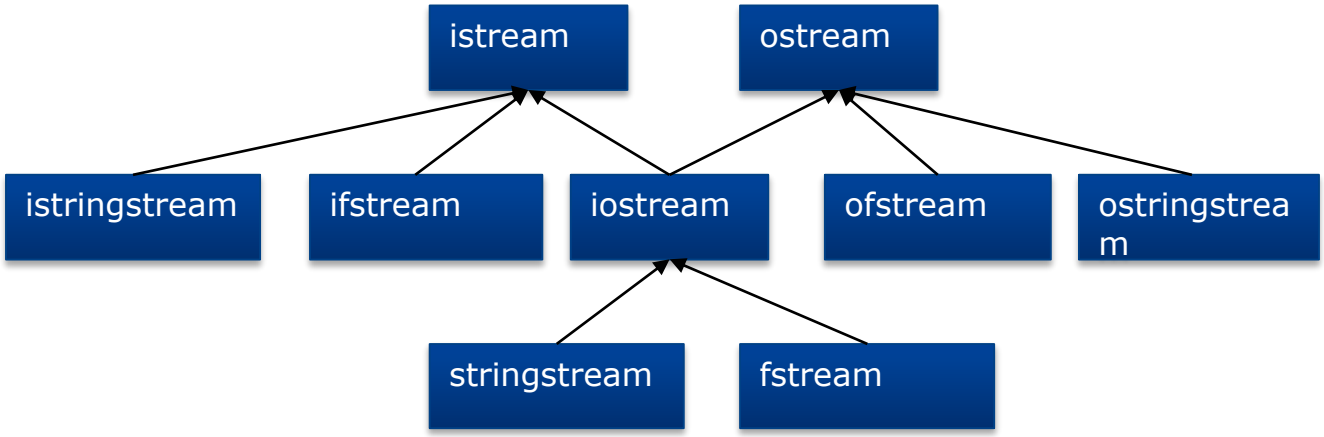
```
template<typename Target, typename Source>
Target lexical_cast(Source arg)
{
    std::stringstream ss;
    Target result;

    if (!(ss << arg)                    // read arg into stream
    || !(ss >> result)                  // read result from stream
    || !(ss >> std::ws).eof())   // stuff left in stream?
            throw bad_lexical_cast();

    return result;
}
```

参考 boost::lexical_cast

```
string s = lexical cast<string>(lexical_cast<double>("   12.7    ")); // ok
// works for any type that can be streamed into and/or out of a string:
XX xx = lexical_cast<XX>(lexical_cast<YY>(XX(whatever)));      // !!!
```

I/O 概览

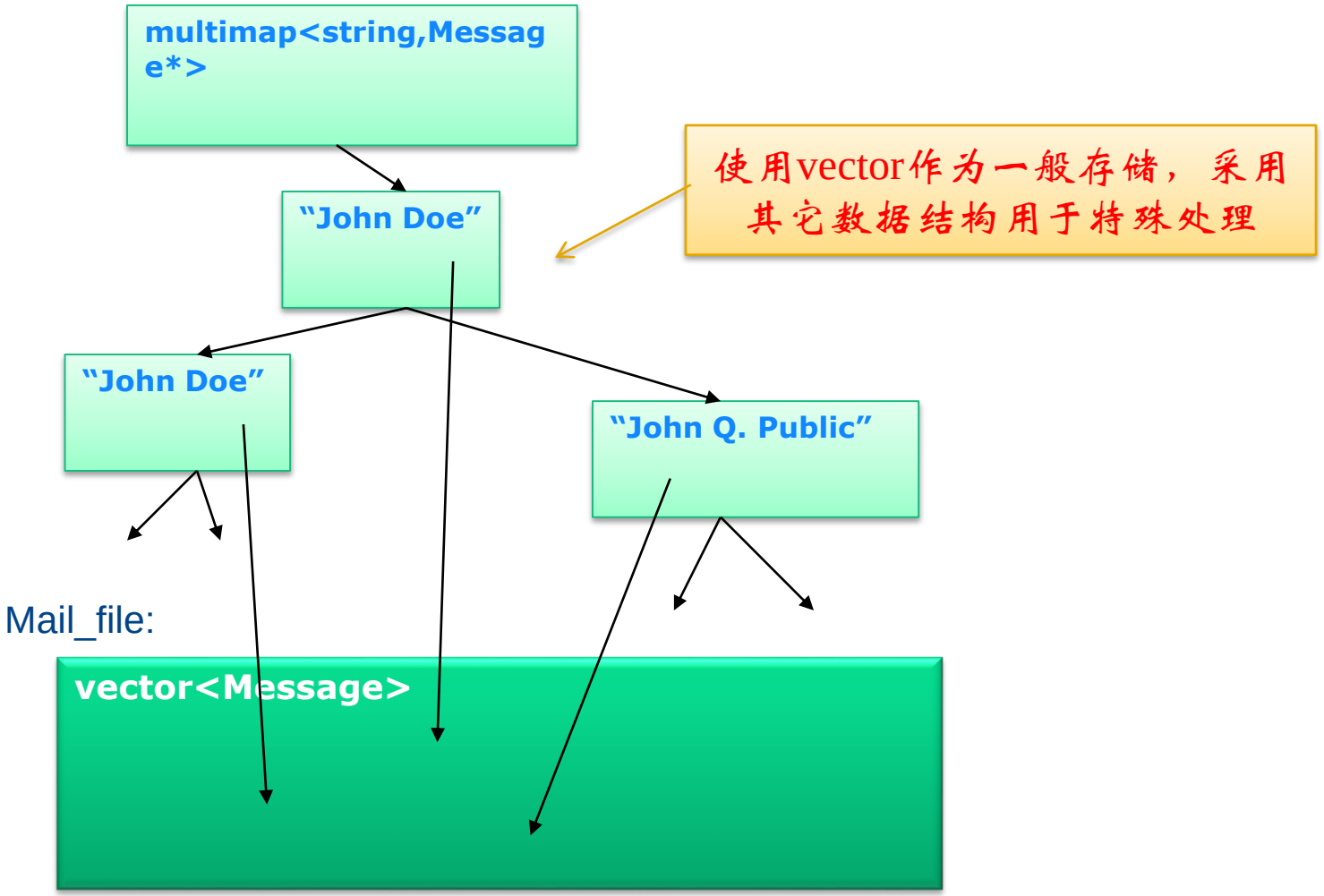
	Stream I/O
in >> x	根据 x 的格式从 in 读取到 x 中
out << x	根据 x 的格式将 x 写入 out 中
in.get(c)	从 in 中读取一个字符到 c 中
getline(in,s)	从 in 中读取一行到字符串 s 中



Map 概览

- 关联容器
  - <map>, <set>, <unordered_map>, <unordered_set>
  - map
  - multimap
  - set
  - multiset
  - unordered_map
  - unordered_multimap
  - unordered_set
  - unordered_multiset
- 文本操作的支柱
  - 找出一个单词
  - 看看是否已经有一个单词
  - 获取给定单词的相关信息
- 具体示例参见 Chapter 23

Map 概览


multimap<string,Message*>
"John Doe"
使用vector作为一般存储，采用其它数据结构用于特殊处理
"John Doe"
"John Q. Public"
Mail_file:
vector<Message>

问题：读取一个邮政编码

- 美国州缩写和 ZIP 码
  - 两个字母紧接着是5个数字

```
string s;
while (cin>>s) {
    if (s.size()==7
    && isletter(s[0]) && isletter(s[1])
    && isdigit(s[2]) && isdigit(s[3]) && isdigit(s[4])
    && isdigit(s[5]) && isdigit(s[6]))
        cout << "found " << s << '\n';
}
```

- 脆弱的、凌乱的、绑定的(不易维护)

问题：读取一个邮政编码

- 上述简单解决方案的问题
  - 它太冗长了(4行代码8个函数调用)
  - 我们忽略了(有意的？)所有没有用空白符将自身与上下文分开的邮政编码
    - “TX77845”、TX77845-1234 和 ATM77845
  - 我们忽略了(有意的？)所有用空白符分割州名缩写和数字的邮政编码
    - TX 77845
  - 我们接受(有意的？)所有使用小写字母的邮政编码
    - tx77845
  - 如果我们确定查找一个不同格式的邮政编码，我们将不得不完全重新代码
    - CB3 0DS, DK-8000 Arhus

TX77845-1234

- 1st 尝试：**wwddddd**
- 2nd (记得 -1234)：**wwddddd-dddd**
- 如何表达“特殊字符”？
- 3rd：**\w\w\d\d\d\d\d-\d\d\d\d**
- 4th (显示计数)：**\w2\d5-\d4**
- 5th (“转义”)：**\w{2}\d{5}-\d{4}**
- 然而 -1234 是可选的？
- 6th：**\w{2}\d{5}(-\d{4})?**
- 我们想在TX后有一个可选的空格
- 7th (看不到的空格)：**\w{2} ?\d{5}(-\d{4})?**
- 8th (空格可视化)：**\w{2}\s?\d{5}(-\d{4})?**
- 9th (0到多个空格)：**\w{2}\s*\d{5}(-\d{4})?**

Regex 库 — 可用

- 它不属于 C++98 标准内
- “Technical Report 1” 2004 的一部分
- 将纳入 C++0x 标准中
- 使用
  - VS 9.0 C++， 使用 **<regex>, std::tr1::regex**
  - GCC 4.3.0， 使用 **<tr1/regex>, std::tr1::regex**
  - www.boost.org， 使用**<boost/regex>, std::boost::regex**

```
#include <boost/regex.hpp>
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

int main()
{
    ifstream in("file.txt");                // input file
    if (!in) cerr << "no file\n";

    regex pat ("\\w{2}\\s*\\d{5}(-\\d{4})?");     // ZIP code pattern
    cout << "pattern: " << pat << '\n';

    // ...
}
```

```
int lineno = 0;
string line;                                    // input buffer
while (getline(in,line)) {
    ++lineno;
    smatch matches;             // matched strings go here
    if (regex_search(line, matches, pat)) {
        cout << lineno << ": " << matches[0] << '\n';    // whole match
        if (1<matches.size() && matches[1].matched)
            cout  << "\t: " << matches[1] << '\n‘; // sub-match
    }
}
```

结果

Input:

address TX77845 1
ffff tx 77843 asasasaa 2
ggg TX3456-23456 3
howdy 4
zzz TX23456-3456sss ggg TX33456-1234 5
cvzcv TX77845-1234 sdsas 6
xxxTx77845xxx 7
TX12345-123456 8

Output:

pattern: "\w{2}\s*\d{5}(-\d{4})?"
1: TX77845
2: tx 77843
5: TX23456-3456
: -3456
6: TX77845-1234
: -1234
7: Tx77845
8: TX12345-1234
: -1234

正则表达式语法

- 正则表达式的完整理论基础是基于状态机的高效实现
  - 你能够干预语法，但不要修改语义
- 语法是简明、神秘和烦人的，但很有用
  - 学习它吧……
- 例如：
  - **Xa{2,3}** // Xaa Xaaa
  - **Xb{2}** // Xbb
  - **Xc{2,}** // Xcc Xccc Xcccc Xccccc …
  - **\w{2}-\d{4,5}** // \w is letter \d is digit
  - **(\d*:)?(\d+)** // 124:1232321 :123 123
  - **Subject: (FW:|Re:)?(.*)** // . (dot) matches any character
  - **[a-zA-Z] [a-zA-Z_0-9]*** // identifier
  - **[^aeiouy]** // not an English vowel

搜索 VS. 匹配

- *搜索 (Searching)* 是在任意数据流中搜索与正则表达式匹配的字符串
  - **regex_search( )** 在流中找出该模式作为子字符串
- *匹配 (Matching)* 是在一个给定大小的字符串中匹配字符串
  - **regex_match( )** 找出模式的完整匹配及其字符串

从 web 上获取的表格

KLASSE		ANTAL DRENGE		ANTAL PIGER	ELEVER IALT
0A	12	11	23		
1A	7	8	15		
1B	4	11	15		
2A	10	13	23		
3A	10	12	22		
4A	7	7	14		
4B	10	5	15		
5A	19	8	27		
6A	10	9	19		
6B	9	10	19		
7A	7	19	26		
7G	3	5	8		
7I	7	3	10		
8A	10	16	26		
9A	12	15	27		
0MO	3	2	5		
0P1	1	1	2		
0P2	0	5	5		
10B	4	4	8		
10CE	0	1	1		
1MO	8	5	13		
2CE	8	5	13		
3DCE	3	3	6		
4MO	4	1	5		
6CE	3	4	7		
8CE	4	4	8		
9CE	4	9	13		
REST	5	6	11		
Alle klasser		184	202	386	

- 数值域
- 文本域
- 不可见的分隔符域
- 语义独立性
  - i.e. 数字实际上表示某种含义
    - first row + second row == third row
    - 最后一行是该列的总和

描述行

第一行

- 正则表达式：^[\w]+([\w]+)*$
- 字符串常量："^[\\w]+([\\w]+)*$"

其他行

- 正则表达式：^([\w]+)(\d+)(\d+)(\d+)$
- 字符串常量："^([\\w]+)(\\d+)(\\d+)(\\d+)$"

那些不可见的tab符号不讨厌吗？

- 定义一个tab符号类

那些不可见的 space 符号不讨厌吗？

- 使用 \s

简单的布局检查

```
int main()
{
    ifstream in("table.txt"); // input file
    if (!in) error("no input file\n");

    string line;      // input buffer
    int lineno = 0;

    regex  header( "^[\\w ]+(          [\\w ]+)*$");                    // header line
    regex  row( "^([\\w ]+)(  \\d+)(      \\d+)(      \\d+)$"); // data line
    // ... check layout ...
}
```

简单的布局检查

```
int main()
{
    // ... open files, define patterns ...

    if (getline(in,line)) {        // check header line
            smatch matches;
            if (!regex_match(line, matches, header)) error("no header");
    }
    while (getline(in,line)) {  // check data line
            ++lineno;
            smatch matches;
            if (!regex_match(line, matches, row))
                    error("bad line", to_string(lineno));
    }
}
```

验证该表

```
int boys = 0;           // column totals
int girls = 0;

while (getline(in,line)) {          // extract and check data
        smatch matches;
        if (!regex_match(line, matches, row)) error("bad line");

        int curr_boy = from_string<int>(matches[2]);    // check row
        int curr_girl = from_string<int>(matches[3]);
        int curr_total = from_string<int>(matches[4]);
        if (curr_boy+curr_girl != curr_total) error("bad row sum");

        if (matches[1]==“Alle klasser”) {     // last line;  check columns:
                if (curr_boy != boys) error("boys don’t add up");
                if (curr_girl != girls) error("girls don’t add up");
                return  0;
        }

        boys += curr_boy;
        girls += curr_girl;
}
```

应用领域

- 文本处理仅仅是众多领域中的一个
  - 或者说其中的若干领域(看你怎么算了)
  - Browsers, Word, Acrobat, Visual Studio, …
- 图形处理
- 声音处理
- 数据基础
  - 医学
  - 科学
  - 商业
  - …
- 数值计算
- 金融
- …