Parasol
Smarter computing.
Texas A&M University

Chapter 23
Text Processing

Bjarne Stroustrup

www.stroustrup.com/Programming

Overview

- Application domains
- Strings
- I/O
- Maps
- Regular expressions

Table 1: Special Characters

Character Name	Symbol	Shortcut
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

Now you know the basics

Parasol
Smarter computing.
Texas A&M University

- Really! Congratulations!
- Don’t get stuck with a sterile focus on programming language features
- What matters are programs, applications, what good can you do with programming
  - Text processing
  - Numeric processing
  - Embedded systems programming
  - Banking
  - Medical applications
  - Scientific visualization
  - Animation
  - Route planning
  - Physical design

Text processing

- “all we know can be represented as text”
  - And often is
- Books, articles
- Transaction logs (email, phone, bank, sales, …)
- Web pages (even the layout instructions)
- Tables of figures (numbers)
- Graphics (vectors)
- Mail
- Programs
- Measurements
- Historical data
- Medical records
- …

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

String overview

Parasol
Smarter computing.
Texas A&M University

- Strings
  - **std::string**
    - **<string>**
    - **s.size()**
    - **s1==s2**
  - C-style string (zero-terminated array of char)
    - **<cstring>** or **<string.h>**
    - **strlen(s)**
    - **strcmp(s1,s2)==0**
  - **std::basic_string<Ch>**, e.g. Unicode strings
    - **using string = std::basic_string<char>;**
  - Proprietary string classes

C++11 String Conversion

Parasol
Smarter computing.
Texas A&M University

- In <string>, for numerical values
- For example:

```
string s1 = to_string(12.333);        // "12.333"
string s2 = to_string(1+5*6-99/7);    // "17"
```

String conversion

- We can write a simple **to_string()** for any type that has a "put to" operator<<

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

- For example:

```
string s3 = to_string(Date(2013, Date::nov, 14));
```

C++11 String Conversion

Parasol
Smarter computing.
Texas A&M University

- Part of <**string**>, for numerical destinations
- For example:

```
string s1 = "-17";
int x1 = stoi(s1);        // stoi means string to int

string s2 = "4.3";
double d = stod(s2);  // stod means string to double
```

String conversion

Parasol
Smarter computing.
Texas A&M University

- We can write a simple **from_string()** for any type that has an "get from" operator<<

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

- For example:

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

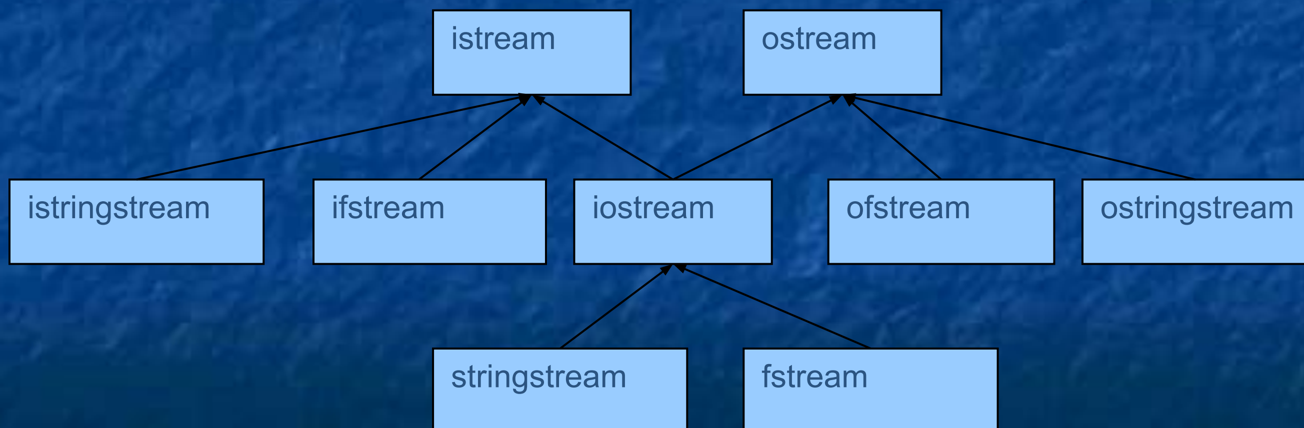
General stream conversion


```
template<typename Target, typename Source>
Target to(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;
}

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

I/O overview

Parasol
Smarter computing.
Texas A&M University

	Stream I/O
in >> x	Read from **in** into **x** according to **x**'s format
out << x	Write **x** to **out** according to **x**'s format
in.get(c)	Read a character from **in** into **c**
getline(in,s)	Read a line from **in** into the string **s**

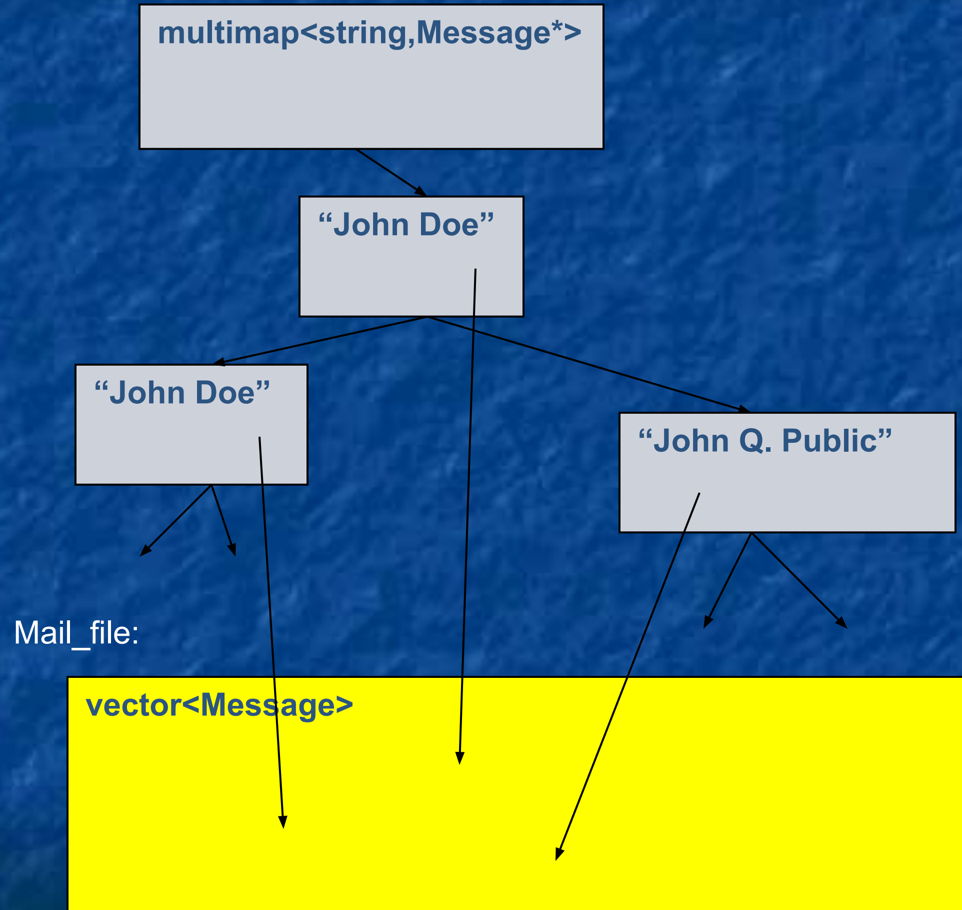


Map overview

Parasol
Smarter computing.
Texas A&M University

- Associative containers
  - **<map>, <set>, <unordered_map>, <unordered_set>**
  - **map**
  - **multimap**
  - **set**
  - **multiset**
  - **unordered_map**
  - **unordered_multimap**
  - **unordered_set**
  - **unordered_multiset**
- The backbone of text manipulation
  - Find a word
  - See if you have already seen a word
  - Find information that correspond to a word
- See example in Chapter 23

Map overview



A problem: Read a ZIP code

Parasol
Smarter computing.
Texas A&M University

- U.S. state abbreviation and ZIP code
  - two letters followed by five digits

```
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';
}
```

- Brittle, messy, unique code

A problem: Read a ZIP code

Parasol
Smarter computing.
Texas A&M University

- Problems with simple solution
  - It's verbose (4 lines, 8 function calls)
  - We miss (intentionally?) every ZIP code number not separated from its context by whitespace
    - **"TX77845"**, **TX77845-1234**, and **ATM77845**
  - We miss (intentionally?) every ZIP code number with a space between the letters and the digits
    - **TX 77845**
  - We accept (intentionally?) every ZIP code number with the letters in lower case
    - **tx77845**
  - If we decided to look for a postal code in a different format we would have to completely rewrite the code
    - **CB3 0DS**, **DK-8000 Arhus**

TX77845-1234

- 1st try: **wwddddd**
- 2nd (remember -12324): **wwddddd-dddd**
- What's "special"?
- 3rd: **\w\w\d\d\d\d\d-\d\d\d\d**
- 4th (make counts explicit): **\w2\d5-\d4**
- 5th (and "special"): **\w{2}\d{5}-\d{4}**
- But -1234 was optional?
- 6th: **\w{2}\d{5}(-\d{4})?**
- We wanted an optional space after TX
- 7th (invisible space): **\w{2} ?\d{5}(-\d{4})?**
- 8th (make space visible): **\w{2}\s?\d{5}(-\d{4})?**
- 9th (lots of space – or none): **\w{2}\s*\d{5}(-\d{4})?**

```
#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';  //   printing of patterns is not C++11

    // …
}
```

Parasol
Smarter computing.
Texas A&M University

```
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
    }
}
```

Results

Input: address TX77845
ffff tx 77843 asasasaa
ggg TX3456-23456
howdy
zzz TX23456-3456sss ggg TX33456-1234
cvzcv TX77845-1234 sdsas
xxxTx77845xxx
TX12345-123456

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

Regular expression syntax

Parasol
Smarter computing.
Texas A&M University

- Regular expressions have a thorough theoretical foundation based on state machines
  - You can mess with the syntax, but not much with the semantics
- The syntax is terse, cryptic, boring, useful
  - Go learn it
- Examples
  - **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

Searching vs. matching

Parasol
Smarter computing.
Texas A&M University

- *Searching* for a string that matches a regular expression in an (arbitrarily long) stream of data
  - **regex_search()** looks for its pattern as a substring in the stream
- *Matching* a regular expression against a string (of known size)
  - **regex_match()** looks for a complete match of its pattern and the string

Table grabbed from the web

Parasol
Smarter computing.
Texas A&M University

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		

- Numeric fields
- Text fields
- Invisible field separators
- Semantic dependencies
  - i.e. the numbers actually mean something
    - first row + second row == third row
    - Last line are column sums

Describe rows

Parasol
Smarter computing.
Texas A&M University

- Header line
  - Regular expression: **^[\w]+(	[\w]+)*$**
  - As string literal: **"^[\\w]+(	[\\w]+)*$"**
- Other lines
  - Regular expression: **^([\w]+)(	\d+)(	\d+)(	\d+)$**
  - As string literal: **"^([\\w]+)(	\\d+)(	\\d+)(	\\d+)$"**
- Aren’t those invisible tab characters annoying?
  - Define a tab character class
- Aren’t those invisible space characters annoying?
  - Use **\s**

Simple layout check

Parasol
Smarter computing.
Texas A&M University

```
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 ...
}
```

Simple layout check

Parasol
Smarter computing.
Texas A&M University

```
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));
    }
}
```

Validate table

```
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;
}
```

Application domains

- Text processing is just one domain among many
  - Or even several domains (depending how you count)
  - Browsers, Word, Acrobat, Visual Studio, …
- Image processing
- Sound processing
- Data bases
  - Medical
  - Scientific
  - Commercial
  - …
- Numerics
- Financial
- Real-time control
- …