Input - Output

- Streams
- File I/O
- String streams

Streams

Streams

- A stream is like a pipe which carries ordered sequence of characters of arbitrary length from a source to a destination
- In a pipe, water enter from one side (the source) and gets out on the other side (the destination)
- Similarly, characters enter in the stream in a certain order and gets out in the same order

cin and cout

- **cin** and **cout**, which we used so far, are streams
- In particular
  - cin is an input stream, where the source is the standard input (normally the keyboard) and the output is controlled by your program

    http://en.cppreference.com/w/cpp/io/cin
  - cout is an output stream (class **ostream**), where the destination is the standard output (normally the console) and the input is controlled by your program

    http://www.cplusplus.com/reference/iostream/cout/
- cin and cout are global instances of classes defined in the STL library (resp. **istream** and **ostream**)

Operator << with streams

- **<<** is a C++ operator, natively defined for integer types, which carries out a binary shift

```
int x = 3;   // in binary representation: 0b0011
c = x << 2;  // shift left-wise by two bits: 0b1100
cout << x << '\n';   // this outputs 12
```

- For the class **ostream** it is overloaded and has a completely different semantic. E.g., for *bool*:

```
ostream& operator << (ostream& os, bool x)
{
   os.put( x? '1' : '0');  // put a character in the stream
   return os;
}
```

Streams is an Abstraction

- The idea of streams is that they are conceptually an abstraction
- Suppose we are writing a function to output a dynamic array to a generic device using streams
- We want to work for arrays of any type T and for any output stream for whom an implementation of **operator<<** exists

Example – Stream Abstraction

```
#include <iostream>

template <typename OS, typename T>
void printArray(OS& os, const T* p, unsigned n)
{
   // for this code to work, we need os << p[i] to be defined, i.e.
   // OS& operator<<(OS& os, const T& v) must exist
   for (unsigned i = 0; i < n; ++i)
     os << p[i] << " ";
   os << std::endl;
}

int main()
{
     int x[] = {1,2,3,4};
     // we could use with something other than cout
     printArray(std::cout, x, 4);
     return 0;
}
```

File I/O

File I/O

- A file is just another stream.
- In the command **cout << x*;*** we just have replace cout with something else.
- Streams are the modern way to do input output
- In C this was done with the CRT library functions *printf* (for writing) and *scanf* (for reading) and related functions.
- The *printf* syntax is still popular. We will not discuss it, but you may want to read about it on your own, together with the CRT functions for string manipulation (e.g. *strelen*, *strcmp*, …)

Output

- Output to file requires an **ofstream**
- File I/O requires `#include<fstream>`
- First we declare a variable of type of

```
ofstream f(“myfile.txt”);
```

- Can use either a **relative** or **absolute** path
- This will create the file myfile.txt, if it does not already exists, or overwrite it, if it already exists
- Now we can write to it, as we would do with cout

```
f << x << endl;
```

Example - Output

```
int main()
{
    // declare and open the stream, associated with a file
    std::ofstream f("myfile.txt");  // relative path

    // write to the stream (i.e. to the file)
    f << "Hello world\n";

    // close the stream (i.e. the file)
    f.close();

    return 0;
}
```

Example – Stream Abstraction

```
#include <fstream>
#include <iostream>
template <typename OS, typename T>

void printArray(OS& os, const T* p, unsigned n)
{
   for (unsigned i = 0; i < n; ++i)
     os << p[i] << " ";    // we need os << p[i] to be defined
     os << std::endl;
}

int main()
{
   int x[] = {1,2,3,4};
   printArray(std::cout, x, 4);  // send to cout
   std::ofstream f("myfile.txt");  // open file and associate stream f
   printArray(f, x, 4);            // send to stream f (i.e. to file)
   return 0;
}  // here f is deleted and the file is closed
```

Output

- What if, we want to append text to an existing file, instead of overriding it?
- We can pass extra arguments to ofstream:

```
std::ofstream f("myfile.txt", std::ios::app );
```

- Let's look at the methods available for *ofstream* and attributes available for *ios*

http://www.cplusplus.com/doc/tutorial/files/

http://www.cplusplus.com/reference/fstream/ofstream/

File Input

- File input needs `#include<fstream>`
- An input stream that reads from the file ***fileName*** is created by

  `ifstream` ***`streamName`*** **`(`*****`fileName`*****`);`**

- After this, ***streamName*** can be used completely similar to `cin`
- ***fileName*** can be an absolute path, e.g. "`Ctest.txt`",
  or a path relative to the directory of the executable C++ program,
  e.g. "`input.txt`"

Useful Functions for File Input

- Assume an input stream in has been declared, e.g. with ifstream in(“input.txt”);
- in.eof() returns true if the ***e**nd **o**f the **f**ile* has been reached, otherwise false
- in.fail() returns true if the last input failed (e.g. string is attempted to be read into an integer), otherwise false
- in.clear() resets the input stream to its original state after an input failed

Using a while-loop for File Input

- Often we do not know how many data values a file contains
- Need to know when the end of the file is reached
- Solution:
  - create a temporary variable, say “buffer”
  - create an input stream, say “in”
  - use while(in>>buffer) or while(!in.eof())
  - in the while-loop, append the values to an array
- Why it works: (in>>buffer) will be false if and only if the end of the input is reached or an input error is encountered

Problem

- Write a program that writes the numbers 1,2, ... ,100 and their square roots to a file SquareRoots.txt
- Try two versions: one with absolute and one with relative path
- Then read them back
- Solution in *ReadWrite.cpp*
- The content of the file should look like this:

```
1 1
2 1.41421
3 1.73205
…
```

Problem

- Create a textfile `input1.txt` with the content shown below
- Read the values contained in the file using a loop while(!in.eof()) and print the entries to the screen

```
2342234.02384
3424.340
340349.29347
923482.23784
92347.347
20342.234234
820482.03284
```

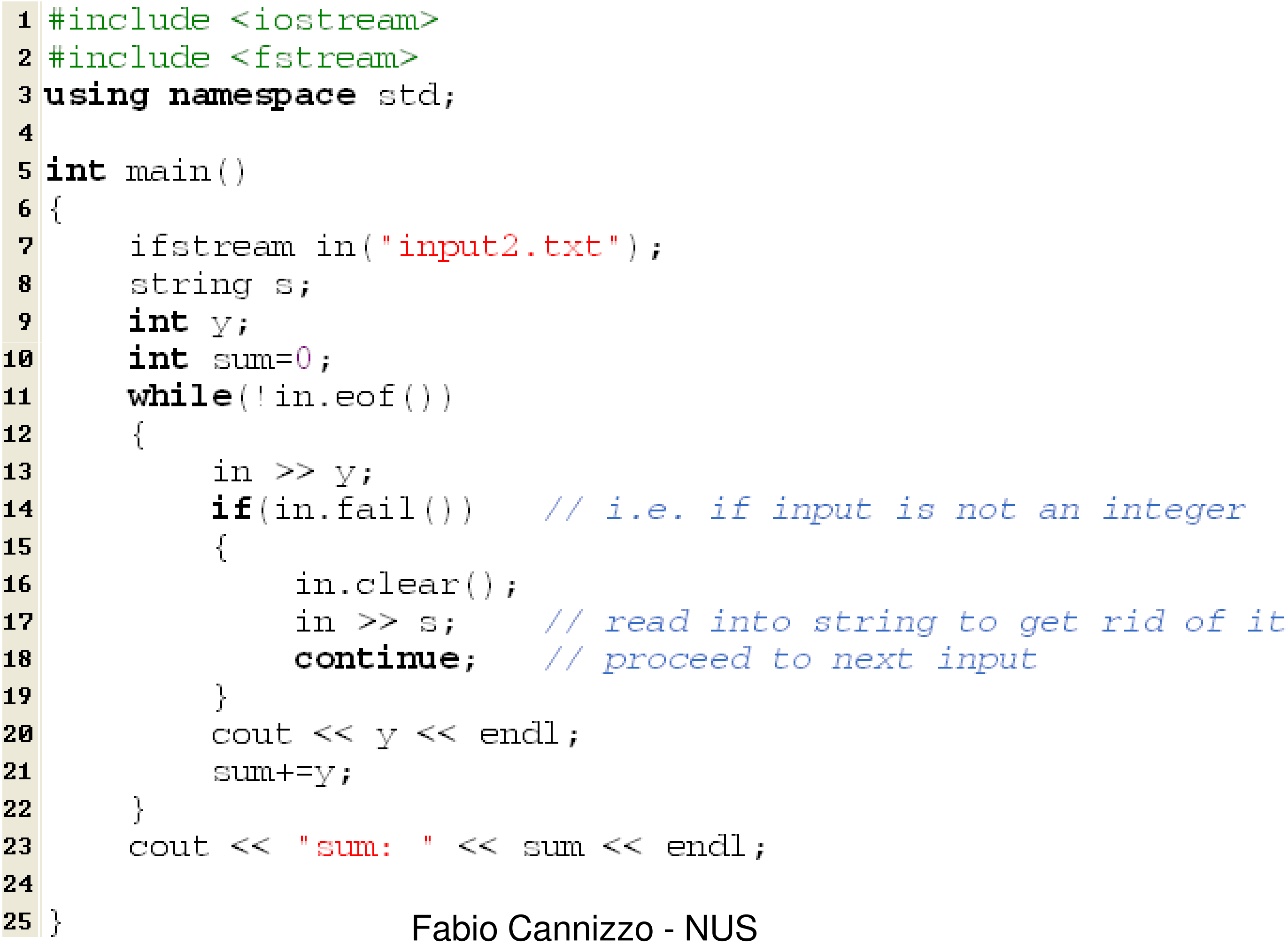
Example 2 (extract numbers from a file)

❑ Create a textfile `input2.txt` with the content shown below

❑ Read the numbers from the file into a C++ program and print their sum to the screen, ignoring invalid numbers

```
jjsf
3444
&*^*&(
3234
abc
def
123
xyz
```

Solution



```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    ifstream in("input2.txt");
    string s;
    int y;
    int sum=0;
    while(!in.eof())
    {
        in >> y;
        if(in.fail())    // i.e. if input is not an integer
        {
            in.clear();
            in >> s;     // read into string to get rid of it
            continue;    // proceed to next input
        }
        cout << y << endl;
        sum+=y;
    }
    cout << "sum: " << sum << endl;

}
```

Problem (Reading Text from File)

- Create a textfile `input3.txt` with this content:

```
This is a textfile
containing text.
```

- Read the symbols in this file and print the symbols read on the screen
- Note that spaces are gone
- To read spaces, try ifstream in(“input3.txt); string s; getline(in, s); etc. as an alternative

Solution 1

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
    ifstream f("input.txt");
    string s;
    while(!f.eof())
    {
       // the operator >> interprets a whitespaces
       // (e.g. spaces, end of lines) as the end of the string.
       // As a result whitespaces are removed
       f >> s;
       cout << s;
    }
    return 0;
}
```

Solution 2

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
    ifstream f("input.txt");
    string s;
    while(!f.eof())
    {
       // getline reads an entire line as it is.
       getline(f,s);
       cout << s << "\n";
    }
    return 0;
}
```

“Messy” Input Files

- Often the data in input files are not arranged in a good way for C++ input
- Still we need to be able to extract the information we need
- Need to read and experiment with the functions available for *ifstream*

Binary Streams

- Streams are sequence of bytes
- They do not need to be necessarily human-readable sequence of characters
- They can be used to save to file data directly in their binary representation
- We just need to open a file in binary format
- For example, if we have *float x=4.3*, we can save to file the three characters ‘4’, ‘.’ and ‘3’, or the 4 bytes of the IEEE-754 binary image
- See ofstream constructor:

  http://www.cplusplus.com/reference/fstream/ofstream/ofstream/

Problem

- Create an array of 10 random int
- Save it to a file “myfile.bin” in binary format
- Read it back and print the numbers to the screen
- Hint: for manipulation of binary files (how to read and write) look at the example in http://www.cplusplus.com/doc/tutorial/files/
- Try to edit “myfile.bin” with a text editor (e.g. notepad), what do you see?