Programming Expertise
IO, Filesytem
and Regular Expressions

Kappel & Groth
2023-06-29

Last week summary

- destructors (virtual destructors if virtual functions)
- static members variables and functions
- const member functions
- inheritance vs composition
- (multiple inheritance)
- (abstract classes)
- std::string
- std::regex
- auto topics

4 Input/ Output and Regular Expressions

Outline

Cpp Reference

C++ reference

C++98, C++03, C++11, C++14, C++17, C++20, C++23 | Compiler support C++11, C++14, C++17, C++20, C++23

- Freestanding implementations

Language
- Basic concepts
- Keywords
- Preprocessor
- Expressions
- Declaration
- Initialization
- Functions
- Statements
- Classes
- Overloading
- Templates
- Exceptions

Headers

Named requirements

Feature test macros (C++20)

Language support library
- Type support – traits (C++11)
- Program utilities
- Coroutine support (C++20)
- Three-way comparison (C++20)
- `numeric_limits` – `type_info`
- `initializer_list` (C++11)

Concepts library (C++20)

Diagnostics library

General utilities library
- Smart pointers and allocators
  - `unique_ptr` (C++11)
  - `shared_ptr` (C++11)
- Date and time
- Function objects – `hash` (C++11)
- String conversions (C++17)
- Utility functions
- `pair` – `tuple` (C++11)
- `optional` (C++17) – `any` (C++17)
- `variant` (C++17) – `format` (C++20) ⇐

Strings library
- `basic_string`
- `basic_string_view` (C++17)
- Null-terminated strings:
  - byte – multibyte – wide

Containers library
- `array` (C++11) – `vector` – `deque`
- `map` – `unordered_map` (C++11)
- `set` – `unordered_set` (C++11)
- `priority_queue` – `span` (C++20)
- Other containers:
  - sequence – associative
  - unordered associative – adaptors

Iterators library

Ranges library (C++20) ⇐

Algorithms library

Numerics library
- Common math functions
- Mathematical special functions (C++17)
- Numeric algorithms
- Pseudo-random number generation
- Floating-point environment (C++11)
- `complex` – `valarray`

Localizations library

Input/output library ⇐
- Stream-based I/O
- Synchronized output (C++20)
- I/O manipulators

Filesystem library (C++17) ⇐

Regular expressions library (C++11)
- `basic_regex` – algorithms ⇐

Atomic operations library (C++11)
- `atomic` – `atomic_flag`
- `atomic_ref` (C++20)

Thread support library (C++11)
- `thread` – `mutex`
- `condition_variable`

Input / Output

- iostream – interacting with the terminal
  - std::cout writing to the terminal
  - std::cin getting input from the terminal
  - std::cerr writing to the error channel immediately
  - std::clog write to the error channel buffered
  - with small programs there is no difference
  - with larger programs and in a pipe difference becomes important
  - example: my latex compile pipeline would stop if I write on cerr ...
- fstream – principle file stream
  - ofstream – output file stream
  - ifstream – input file stream

Opening a file

Syntax:

```
void open(const char *filename,
    ios::openmode mode);
```

flags:

- ios::app – Append mode. All output to that file to be appended to the end.
- ios::ate – Open a file for output and move the read/write control to the end of the file.
- **ios::in** – Open a file for reading.
- **ios::out** – Open a file for writing.
- ios::trunc – If the file already exists, its contents will be truncated before opening the file.

Input / Output example

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

int main () {
   string data ; // changed in 2022
   // open a file in write mode.
   ofstream outfile;
   outfile.open("afile.dat");

   cout << "Writing to the file" << endl;
   cout << "Enter your name: ";
   cin >> data;
```

```
// write inputted data into the file.
outfile << data << endl;

cout << "Enter your age: ";
cin >> data; // or std::getline(cin,data);
cin.ignore();

// again write inputted data into the file.
outfile << data << endl;

// close the opened file.
outfile.close();

// open a file in read mode.
ifstream infile;
infile.open("afile.dat");
```

```
    cout << "Reading from the file" << endl;
    infile >> data;

    // write the data at the screen.
    cout << data << endl;

    // again read the data from the file and display it.
    infile >> data;
    cout << data << endl;

    // close the opened file.
    infile.close();

    return 0;
}
```

```
$ g++  -o inout.cpp.bin -std=c++17 -fconcepts inout.cpp && ./inout.cpp.bin
```

```
Writing to the file
Enter your name: Enter your age: Reading from the file
C++
40
```

stringstream

"A stringstream associates a string object with a stream allowing you to read from the string as if it were a stream (like cin). To use stringstream, we need to include sstream header file. The stringstream class is extremely useful in parsing input."
https://www.geeksforgeeks.org/stringstream-c-applications/

```
#include <bits/stdc++.h>
int main() {
    std::string S, T;   S = "Hello C++ World!";
    std::stringstream X(S);
    while (std::getline(X,T,' ')){ std::cout << T << std::endl; }
    return 0;
}
```

$ g++ -o sstream.cpp.bin -std=c++17 -fconcepts sstream.cpp && ./sstream.cpp.bin

```
Hello
C++
World!
```

Methods of file channels

- open
- close
- getline
- seekg – go into channel position
- tellg – what is the byte position

$\Rightarrow$ You should have in your snippets.conf snippets “openr” and “openw”

IMatrix example - Tabfile reading

```
class IMatrix {
    typedef std::vector<std::string> names ;
    typedef std::vector< std::vector<int> > IntMatrix;
    ...
  void readTabfile (std::string filename) {
     names rnms ;
     names cnms ;
     std::string rname, cname, line;
     row r;
     int rownumber = 0;
     std::string scell;
     int icell;
     // open a file in read mode.
     std::ifstream infile;
     infile.open(filename);
```

```
IntMatrix tmt;
while (std::getline(infile, line)) {
    rownumber += 1;
    std::istringstream iss(line);
    if (rownumber == 1) {
        iss >> scell ; // skip rownames entry
        while (iss >> scell) {
            cnms.push_back(scell);
        }
    } else {
        iss >> scell ;
        r.clear();
        rnms.push_back(scell);
        while ((iss >> icell)) {
            r.push_back(icell);
        }
```

```
            tmt.push_back(r);
        }
    }
    infile.close();
    //return;
    this->rnames=rnms;
    this->cnames=cnms;
    this->mt=tmt;
    std::cout << "done" << std::endl;
}
```

IMatrix example - Tabfile writing

```
void writeTabfile (std::string filename) {
    names cnms, rnms ;
    if (cnames.size()==0) {
        cnms=autonames(ncol(),"C");
    } else {
        cnms=cnames;
    }
    if (rnames.size()==0) {
        rnms=autonames(nrow(),"R");
    } else {
        rnms=rnames;
    }
    std::ofstream outfile;
    outfile.open(filename,
       std::ios::out | std::ios::trunc );
```

```
    outfile << "RowNames" ;
    for (auto c : cnms) {
        outfile << "\t" << c ;
    }
    outfile << "\n" ;
    int x = 0;
    for (auto r : mt) {
        outfile << rnms[x++] ;
        for (auto c : r) {
            outfile << "\t" << c ;
        }
        outfile << "\n" ;
    }
    outfile.close();
}
```


IMatrixC.cpp

Filesystem

$\Rightarrow$ introduced in C++17 and were taken from the Boost libraries

Core parts:

- path object
- directory_entry
- directory iterators
- supportive functions
  - information about the path
  - file manipulations (copy, move, create, ...)
  - file properties, time, size, ...
  - ...

Filesystem API

Filesystem library

Classes

filesystem::path
filesystem::filesystem_error
filesystem::directory_entry
filesystem::directory_iterator
filesystem::recursive_directory_iterator
filesystem::file_status
filesystem::space_info
filesystem::file_type
filesystem::perms
filesystem::perm_options
filesystem::copy_options
filesystem::directory_options
filesystem::file_time_type

Functions

filesystem::absolute
filesystem::canonical
filesystem::weakly_canonical
filesystem::relative
filesystem::proximate
filesystem::copy
filesystem::copy_file
filesystem::copy_symlink
filesystem::create_directory
filesystem::create_directories
filesystem::create_hard_link
filesystem::create_symlink
filesystem::create_directory_symlink
filesystem::current_path
filesystem::exists
filesystem::equivalent
filesystem::file_size
filesystem::hard_link_count
filesystem::last_write_time
filesystem::permissions
filesystem::read_symlink
filesystem::remove
filesystem::remove_all
filesystem::rename
filesystem::resize_file
filesystem::space
filesystem::status
filesystem::symlink_status
filesystem::temp_directory_path

File types

filesystem::is_block_file
filesystem::is_character_file
filesystem::is_directory
filesystem::is_empty
filesystem::status_known
filesystem::is_fifo
filesystem::is_other
filesystem::is_regular_file
filesystem::is_socket
filesystem::is_symlink

Filesystem example

```
#include <iostream>
#include <version>

// check for older compilers
#ifdef __cpp_lib_filesystem
    #include <filesystem>
    namespace fs = std::filesystem;
#elif __cpp_lib_experimental_filesystem
    #include <experimental/filesystem>
    namespace fs = std::experimental::filesystem;
#else
    #error "no filesystem support ='("
#endif

int main () {
  std::cout << "current_path:\n" << fs::current_path() <<
```

```
  "\nExists Makefile?: " << fs::exists("../Makefile") <<
    std::endl;
  return(0);
}
```

```
$ g++   -o fsystem.cpp.bin -std=c++17 -fconcepts fsystem.cpp &&
./fsystem.cpp.bin
current_path:
"/home/groth/workspace/delfgroth/docs/lehre/SS2023/PEX/build"
Exists Makefile?: 1
```

Traversing a directory

⇒ take example DisplayDirTree bfilipek.com

```
#include <iostream>
#include <filesystem>
namespace fs = std::filesystem;

void DisplayDirTree(const fs::path& pathToShow,
    int level) {
  if (fs::exists(pathToShow) &&
    fs::is_directory(pathToShow)) {
    auto lead = std::string(level * 3, ' ');
    for (const auto& entry :
      fs::directory_iterator(pathToShow)) {
      auto filename = entry.path().filename();
    if (fs::is_directory(entry.status())) {
      std::cout << lead << "[+] " << filename << "\n";
```

```
      DisplayDirTree(entry, level + 1);
      std::cout << "\n";
    }
    else if (fs::is_regular_file(entry.status()))
      std::cout << filename << std::endl;
    else
        std::cout << lead << " [?]" << filename << "\n";
    }
  }
}
int main () {
    std::cout << "Directory test:\n";
    DisplayDirTree("test",0);
    return(0);
}
```

$ g++ -o dir.cpp.bin -std=c++17 -fconcepts dir.cpp && ./dir.cpp.bin

```
Directory test:
"hello.txt"
"hello2.txt"
```

C++17/C++20 code for C++11/C++14

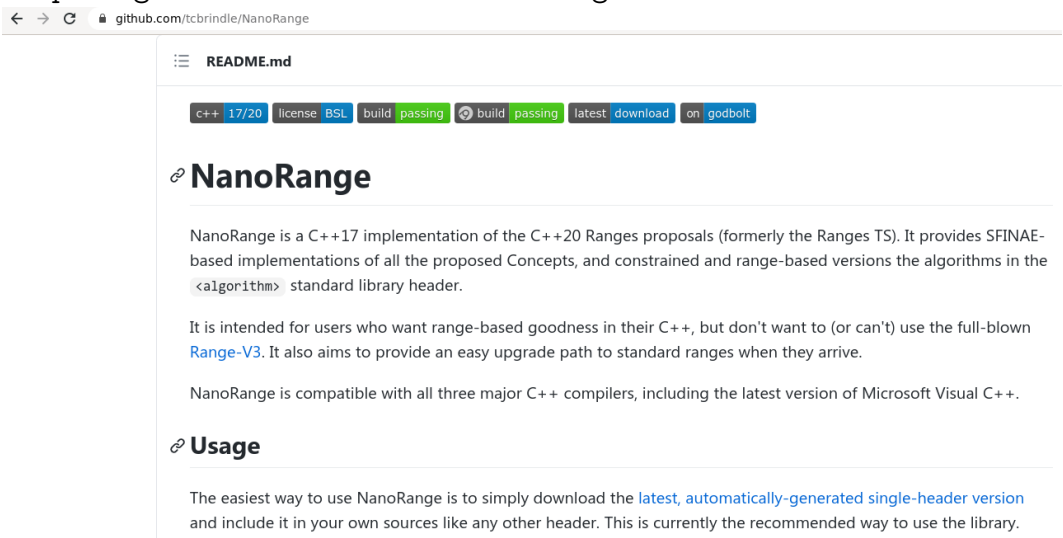
ghc::filesystem for C++11/C++14

Header-only single-file as as a std::filesystem compatible library implemented for C++11 and C++14 `https://github.com/gulrak/filesystem`.

```
#ifdef __cpp_lib_filesystem
    #include <filesystem>
    namespace fs = std::filesystem;
#elif __cpp_lib_experimental_filesystem
    #include <experimental/filesystem>
    namespace fs = std::experimental::filesystem;
#else
    #include "ghc/filesystem.hpp"
    namespace fs = ghc::filesystem;
#endif
```

C++20 Ranges vor C++17

https://github.com/tcbrindle/NanoRange



STL example - C++17 nanorange

```
#include <algorithm>
#include <functional>
#include <array>
#include <iostream>
#include "include/nanorange.hpp"
namespace ranges = nano::ranges;
int main() {
    std::array<int, 10> s = {5, 7, 4, 2, 8, 6,
                             1, 9, 0, 3};
    for (auto a : s) {
        std::cout << a << " ";
    }
    std::cout << '\n';
    // c++17 with nano::ranges
    ranges::sort(s);
```

```
    for (auto a : s) {
        std::cout << a << " ";
    }
    std::cout <<'\n';
    ranges::reverse(s);
    for (auto a : s) {
        std::cout << a << " ";
    }
    std::cout << '\n';

}
```

```
$ g++   -o stl-nano.cpp.bin -std=c++17 -fconcepts stl-nano.cpp &&
./stl-nano.cpp.bin
5 7 4 2 8 6 1 9 0 3
0 1 2 3 4 5 6 7 8 9
9 8 7 6 5 4 3 2 1 0
```

C++17/20 (nano)ranges pipes

```
#include <iostream>
#include <vector>
#include "include/nanorange.hpp"
namespace ranges = nano::ranges;
namespace views = nano::views;
int main() {
    std::vector<int> numbers = {1, 2, 3, 4, 5, 6};
    auto results = numbers | views::filter([](int n){
            return n % 2 == 0; })
                           | views::transform([](int n){
            return n * 2; });
    for (auto v: results) std::cout << v << " ";  // 4 8 12
}
```

$ g++ -o stl-nano2.cpp.bin -std=c++17 -fconcepts stl-nano2.cpp && ./stl-nano2.cpp.bin

```
4 8 12
```

C++20 std::span via nonstd::span

span-lite, optional-lite etc: https://github.com/martinmoene

```
#include <iostream>
#include "include/nonstd/span.hpp"


template<typename T, std::size_t length>
void print(nonstd::span<T, length> span) {
    for(auto i : span) {
        std::cout << i << ' ';
    }
    std::cout << '\n';
}


int main() {
```

```
    int data[] = {1, 2, 3, 4, 5, 6}; // C-array!!
    auto span = nonstd::span{data};
    print(span); // no length required
    print(span.first(3));
    print(span.last(3));
    print(span.subspan(1, 3));
}
```

```
$ g++  -o span.cpp.bin -std=c++17 -fconcepts span.cpp && ./span.cpp.bin
1 2 3 4 5 6
1 2 3
4 5 6
2 3 4
```

A span<T> is a light-weight wrapper around a C-style array, preferred by C++ developers whenever they are using C libraries and want to wrap them with a C++-style data container for "type safety" and "C++-ishness" ...

C++20 std::format via fmt

https://github.com/fmtlib/fmt

```
sudo dnf install fmt-devel
```

```
#include <string>
#include <fmt/core.h>

int main() {
    fmt::print("Hello, world!\n");
    std::string s = fmt::format("The answer is {}.\n", 42);
    fmt::print(s);
    s = fmt::format("I'd rather be {1} than {0}.",
        "right", "happy");
    fmt::print(s);
}
```

```
$ g++  -L. -lfmt -Iinclude -o fmt.cpp.bin fmt.cpp && ./fmt.cpp.bin
Hello, world!
The answer is 42.
I'd rather be happy than right.
```

std::regex - C++11

Regular expressions library

Classes
basic_regex (C++11)
sub_match (C++11)
match_results (C++11)

Algorithms
regex_match (C++11)
regex_search (C++11)
regex_replace (C++11)

Iterators
regex_iterator (C++11)
regex_token_iterator (C++11)

Exceptions
regex_error (C++11)

Traits
regex_traits (C++11)

Constants
syntax_option_type (C++11)
match_flag_type (C++11)
error_type (C++11)

Regex Grammar
Modified ECMAScript-262 (C++11)

```
#include <iostream>
```

```
#include <string>
#include <regex>

int main () {
   std::string s ("remove me please Hello      World");
   // std::string s("string");
   // std::string s = "string";
   // std::string s {"string"};  // preferred way(?)
   std::regex  r (".+(Hello) +(World)");
   std::cout << std::regex_replace(s,r,"$1 $2!\n");
   std::cout << std::regex_replace("hw!\n",
       std::regex("hw"),"Hello World II");
   if (std::regex_search(">id test\nMALF",
        std::regex("^>[^\\s]+") )){
      std::cout << "FASTA header searched\n"; // YES
   }
```

```
    if (std::regex_match(">id test\nMALF",
              std::regex("^>[^\\s]+") )){
       std::cout << "FASTA header matched I\n";  // NO
    }
    if (std::regex_match(">id test\nMALF",
         std::regex("^>[^\\s]+.+\n.+") )){
       std::cout << "FASTA header matched II\n";  // YES
    }
    return 0;
}
```

```
$ g++   -o regex.cpp.bin -std=c++17 -fconcepts regex.cpp &&
./regex.cpp.bin
Hello World!
Hello World II!
FASTA header searched
FASTA header matched II
```

Regex Syntax I

- characters:
  - exact matches:
    x q ATG SS2010
- metacharacters:
  - characters with special meaning:
    [ ] . ? * + | { } () \ ^ $
- metasymbols:
  - sequences of characters with special meaning:
    \s (space character) \t (tab stop) \n (newline) \w (word character)
  - wordcharacter a-z, A-Z, 0-9, including the _ (underscore)

Regex Syntax II

The terminal tool grep:

```
$ grep -E "regex" filename(s)
$ egrep "regex" filename(s)
```

- *regex* regular expression (pattern)
- switch *-E* allows extended mode with regular expressions
- quotes protect special characters like pipes (|)
- *egrep* is like an alias for *grep -E*

Regex Syntax III

- "A"... standard IUPAC one-letter codes
- "." position where any aa is accepted
- "[ALT]" ambiguity any of Ala or Leu or Thr
- "[^AL]" negative ambiguity any aa but not! Ala or Leu.
- ".{2,4}", "L{3}" repetition. two to four amino acids of any type, exactly three Leu
- "L{2,4}" two to four Leu, why not :)
- "+" one or more, X+ == X{1,}
- "*" zero or more, X* == X{0,}
- "?" zero or one, X? == X{0,1}
- "^M" N terminal Met
- "A$" C terminal Ala

Regex Example Patterns

```
[AC].V.{4}[^ED]
This pattern is translated as:
[Ala or Cys]-any-Val-any-any-any-any-
 {any but Glu or Asp}

^A.[ST]{2}.{0,1}V
This pattern, which must be in the
 N-terminal of the sequence (`<'),
is translated as:
Ala-any-[Ser or Thr]-[Ser or Thr]-
  (any or none)-Val
```

```
^[^C]+$
This pattern describes all sequences
which do not contain any Cysteines.

IIRIFHLRNI
This pattern describes all sequences
which contain the subsequence 'IIRIFHLRNI'.
```

Regex in R

- grep: search pattern with index
- grepl: return TRUE or FALSE if pattern is there
- gsub: search pattern and replace with string

```
> grep("A.T",c("AFFT","CCCC","AFT"))
[1] 3
> grepl("A.T",c("AFFT","CCCC","AFT"))
[1] FALSE FALSE  TRUE
> any(grepl("A.T",c("AFFT","CCCC","AFT")))
[1] TRUE
> grep("A.{2}T",c("AFFT","CCCC","AFT"))
[1] 1
> grep("A.{2}T",c("AFFT",'GCFG','TTTTA','GAAATF'))
[1] 1 4
```

```
> gsub("Hallo","Hi","Hallo old world. Hallo means Hey!")
[1] "Hi old world. Hi means Hey!"
> gsub("A.{2}T",'AXXT',c("AFFT",'GCFG','TTTTA','GAAATF'))
[1] "AXXT"   "GCFG"   "TTTTA"  "GAXXTF"
> gsub(">([^\\s]+) .+","\\1",">id1 and some comment")
[1] "id1 and"
```

R vs C++

- R: grepl(pattern,string,flag(s)) - TRUE/FALSE
- C++: regex_search(string, pattern,flag(s))
- Python: re.search(pattern,string,flags(s))
- R: grep(pattern,string) - INDEX (as well vectors)
- C++: not easily possible but see below
- R: gsub(pattern,replace,string)
- C++: regex_replace(string, pattern, replace)
- R replacer: \\N (N: 1-9)
- C++ replacer: $N (N: 1-9)
- Python: re.sub(pattern,replace,string)

replace, match and search

- regex_replace - do substitutions
- regex_match - the complete string must be matched by pattern
- regex_search - the pattern can be somewhere in the string
- match: "^.*<pattern>.*$" == search: "<pattern>"

grep in C++

```
#include <iostream>
#include <vector>
#include <regex>
#include <string>
namespace dutils {
```

```
// a C++ grep which works like the R grep
std::vector<int> grep (std::string pattern, std::string str,
  const std::regex::flag_type & flag = std::regex::basic) {
  std::regex rx;
  rx =std::regex(pattern,flag);
  std::vector<int> index_matches; // results saved here
  for(auto it =
      std::sregex_iterator(str.begin(), str.end(), rx);
      it != std::sregex_iterator(); ++it)  {
          index_matches.push_back(it->position());
  }
  return(index_matches);
}
```

```
std::vector<int> grep (std::string pattern,
    std::vector<std::string> vstring,
    const std::regex::flag_type & flag = std::regex::basic) {
    std::regex rx;
    rx =std::regex(pattern,flag);
    std::vector<int> index_matches = {}; // results saved here
    int i = 0;
    for (auto el : vstring) {
        if (std::regex_search(el,rx)) {
            index_matches.push_back(i);
        }
        i=i+1;
    }
    return(index_matches);
}
} // END OF NAMESPACE
```

```
int main (int argc, char ** argv) {
  std::vector<int> res = dutils::grep("[Hh][ea]",
            "Hello and hallo world!");
   for (auto r : res)
      std::cout << r << std::endl;
   for (auto i :
       dutils::grep("H[ea]",{"Hello","World!",
                "Hallo","Welt!","by","hallo"})) {
       std::cout << i << std::endl;
   }
   for (auto i : dutils::grep("H[ea]",{"Hello","World!",
                              "Hallo","Welt!","by","hallo"},
                             std::regex::icase)) {
        std::cout << i << std::endl;
   }
}
```

$ g++ -o grep.cpp.bin -std=c++17 -fconcepts grep.cpp && ./grep.cpp.bin

```
0
10
0
2
0
2
5
```

Overloading again ... R

Above: C++ grep for string, C++ grep for vector

```
# mean for matrices and vectors
my.mean = function (x) {
    if (is.matrix(x)) {
        res=c()
        for (i in 1:ncol(x)) {
           res=c(res,my.mean(x[,i]))
        }
        return(res)
    } else {
        return(sum(x)/mean(x))
    }
}
```

Overloading again ... C++

```
#include <iostream>
#include <vector>
#include <numeric>
template <typename T>
double mean (std::vector<T> x) {
    double sum = std::accumulate(x.begin(), x.end(), 0);
    return(sum/x.size());
}
template <typename T>
std::vector<double> mean (std::vector< std::vector<T> > x) {
    std::vector<double> res ;
    for (auto vec : x) {
        res.push_back(mean(vec));
    }
    return(res);
}
```

```
int main (int argc, char ** argv) {
    typedef std::vector<int> Vector ;
    typedef std::vector<std::vector<int>> Matrix ;
    Vector x = {1,2,3,4,5,6,10};
    Matrix M =
        { {0,1,2,6},
          {4,5,6,7},
          {8,9,10,19}
        };
    auto xm = mean(x);
    std::cout << "Mean of vector: " << xm << std::endl;
    auto ms = mean(M);
    for (auto v : ms) {
        std::cout << "Mean of column: " << v << std::endl;
    }
}
```

```
$ g++   -o mean.cpp.bin -std=c++17 -fconcepts mean.cpp &&
./mean.cpp.bin
Mean of vector: 4.42857
Mean of column: 2.25
Mean of column: 5.5
Mean of column: 11.5
```

Exercise I

- create your personal namespace like xyutils where xy is your prefix
- implement as well *xyutils::grepl* for strings and vectors returning boolean or vector of booleans
- implement as well *xyutils::any* to check if in a vector any value is true

Homework I

- implement as well *xyutils::gsub* for strings and vectors return single string or vector of strings with replacements

Iterator and Matches

https://www.tutorialspoint.com/cpp_standard_library/cpp_regex_iterator.htm

```
#include <regex>
#include <iterator>
#include <iostream>
#include <string>

int main() {
   const std::string s = "Tutorialspoint.com india pvt ltd.";

   std::regex words_regex("[^\\s]+");
   auto words_begin =
      std::sregex_iterator(s.begin(), s.end(), words_regex);
   auto words_end = std::sregex_iterator();
```

```
    std::cout << "Found "
       << std::distance(words_begin, words_end)
       << " words:\n";
    for (std::sregex_iterator i = words_begin;
     i != words_end; ++i) {
       std::smatch match = *i; // a match object
       std::string match_str = match.str();
       std::cout << match_str << '\n';
    }
}
```

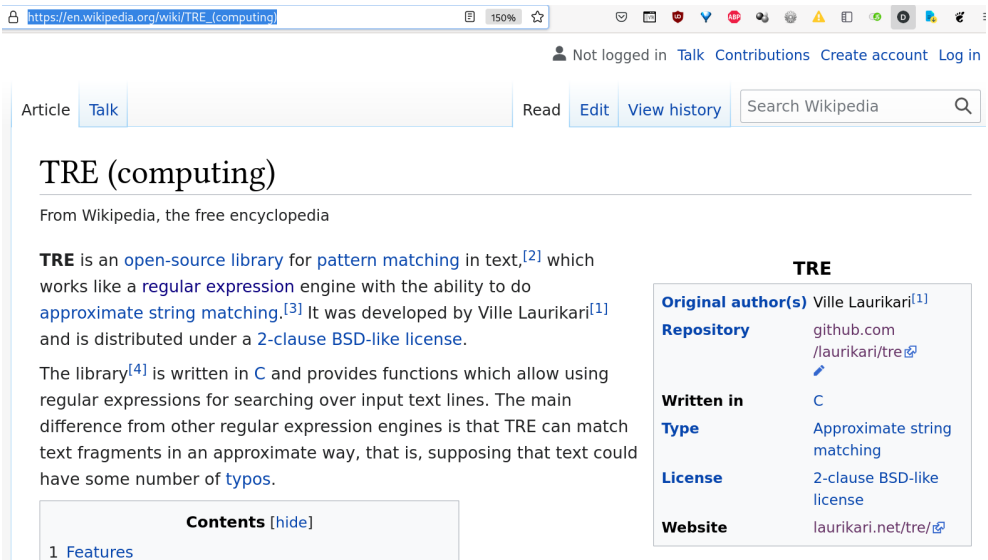
```
$ g++   -o iterator.cpp.bin -std=c++17 -fconcepts iterator.cpp &&
./iterator.cpp.bin
Found 4 words:
Tutorialspoint.com
india
pvt
ltd.
```

Using Regular Expressions on files

Example: open a FASTA file and give out every sequence with its length

```
[groth@bariuke build]$ ./a.out ../../data/human-tRNAs.fasta  | head -n
Given file../../data/human-tRNAs.fasta does  exists!
Scanning ... ../../data/human-tRNAs.fasta!
Homo_sapiens_chr6.trna95-AlaAGC 73
Homo_sapiens_chr6.trna25-AlaAGC 73
Homo_sapiens_chr1.trna87-AlaAGC 71
Homo_sapiens_chr6.trna94-AlaAGC 73
```

TRE: approx Regex (Levenshtein)



https://en.wikipedia.org/wiki/TRE_(computing)

Not logged in Talk Contributions Create account Log in

Article Talk | Read Edit View history | Search Wikipedia

TRE (computing)

From Wikipedia, the free encyclopedia

TRE is an open-source library for pattern matching in text,[2] which works like a regular expression engine with the ability to do approximate string matching.[3] It was developed by Ville Laurikari[1] and is distributed under a 2-clause BSD-like license.

The library[4] is written in C and provides functions which allow using regular expressions for searching over input text lines. The main difference from other regular expression engines is that TRE can match text fragments in an approximate way, that is, supposing that text could have some number of typos.

Contents [hide]

1 Features

TRE

Original author(s)	Ville Laurikari[1]
Repository	github.com /laurikari/tre
Written in	C
Type	Approximate string matching
License	2-clause BSD-like license
Website	laurikari.net/tre/

Regex Links

- Terminal: `Rdoc regex`
- `https://www.tutorialspoint.com/cpp_standard_library/regex.htm`
- `https://github.com/p-ranav/learn-regex`
- `https://dev.to/visheshpatel/introduction-to-regular-expression-with-modern-c-1`
- `https://www.tutorialspoint.com/python/python_reg_expressions.htm`

Summary

- input, output
- std::filesystem
- C++20 features for C++17
- regular expressions
- loop over files
- text substitutions, extractions
- regex_match vs regex_search
- regex_replace
- sregex_iterator and smatch

Exercise IO and Regular Expressions

The actual exercise will be on the Moodle later!

Just suggestions:

Homework (finsh at home):

- create your personal namespace like xyutils where xy is your prefix
- implement as well *xyutils::grepl* for strings and vectors returning boolean or vector of booleans
- implement as well *xyutils::any* to check if in a vector any value is true
- compare it with *all_of* and *any_of* and *none_of* from the standard C++ library `https://en.cppreference.com/w/cpp/algorithm/all_any_none_of`
- implement as well *xyutils::gsub* for strings and vectors return single string or vector of strings with replacements

Homework task with Points

Write an application which gets an FASTA file and a regular expresssion as input and which shows all sequence ids which match this pattern. The pattern should as well work over several lines. Hint: before you apply the pattern you should collect the sequence in one long string.
Do the usual argument checks, file exists checks, etc. As a starter you can have a look at the file *fasta-scan.cpp* on Moodle.

References