SeqAn3 - Sneak Preview

Hannes Hauswedell, Svenja Mehringer, Rene Rahn

September 25, 2018

Freie Universität Berlin, MPI Molekulare Genetik

Overview

Introduction to SeqAn3

Small Read Mapper

Outlook

Introduction to SeqAn3

From SeqAn2 to SeqAn3

Why SeqAn? Why C++?

- ”Don’t pay for what you don’t use”
- If possible use compile-time computation
- Generic algorithms, code re-use
- scale well to cope with large biological datasets
- many existing libraries to combine with

Pitfalls of SeqAn2

- Ahead of its time (good, but resulted in duplication with current Standard)
- ”template subclassing” forced reinvention of many standard data types (e.g. own vector)
- over-verbose use of templates (necessary in C++98)
- incohrent use of design-patterns
- C-like free/global function interface → unintuitive for programmers used to OOP (also no return values, use of out-parameters)
- incomprehensible error messages due to template voodoo

→ steep learning curve

SeqAn3 promises

- easier to learn
- readable error messages
- highly interoperable with the standard library and custom types
- less templates (at least less visible)
- higher code quality and smaller code-base
- at least as fast as before
- BUT probably not the exact same set of features

SeqAn3 technology choices

- use C++17, C++ Concepts and C++ Ranges
- use standard library where possible
- fully integrate the SDSL (succinct datastructure library)
- use existing tooling and drop custom solutions
- stay header-only and BSD-licensed (also dependencies)

Platform support:

- currently only GCC7 and GCC8
- experimental Clang support on the way
- MSVC support likely not before C++20 (in 2020)

Introduction to SeqAn3

C++ Concepts and C++ Ranges

Modern C++ – Concepts

SeqAn2-style generic programming:

```
template <typename TValue, typename TSpec>
void print_first(String<TValue, TSpec> const & s)
{
    std::cout << *begin(s) << '\n';
}
```

SeqAn3-style generic programming with Concepts:

```
template <std::InputRange rng_t>
void print_first(rng_t && s)
{
    debug_stream << *begin(s) << '\n';
}
```

What is `std::InputRange`?

- It's a "concept", i.e. a set of requirements that a type needs to fulfill so that it is accepted as a template argument.
- Requirements can be e.g. "has member function `.x()`".
- **The type doesn't need to know about a a concept to satisfy it.** (in contrast to "interfaces").
- Concepts can be more or less specialised if one concept's requirements are a subset of the other one's; this means concepts can fully replace template-specialisation and/or inheritance as a mechanism for specialisation while seemlessly integrating with external types!

What is `std::InputRange` concretely?

- `std::InputRange` describes types over whom you can iterate from beginning to end by calling `begin()` and `end()` respectively; and whose iterators can be dereferenced.
- Types that satisfy this are e.g. ‘`std::vector<int>` or `std::string`

Modern C++ – Ranges

Why speak of ”ranges” and not ”containers”?

- In modern C++ there is the paradigm of ”ranges” which is more abstract than a ”container”.
- A ”container” makes assumptions about storage and ownership, a range could be anything that you can iterator over.

What types of ranges are commonly used (in SeqAn3)?

1. Containers: have ownership of the elements; e.g. `std::vector`, `std::string`, `seqan3::bitcompressed_vector`
2. Views: lazy-evaluated stateful algorithms on another range that also appear as a range (!); e.g. `view::reverse`, `seqan3::complement`.
3. Adaptations of streams and stream-like objects: An input stream can be modeled as a single-pass input range; e.g. `seqan3::sequence_file_input`, `seqan3::aligment_file_output`

Modern C++ – Ranges – Views

- Views don't own anything, they contain only reference/pointer to another range and "algorithm", can copied cheaply
- views can be combined via piping, like on the unix command line, but they are lazy-evaluated when you access them

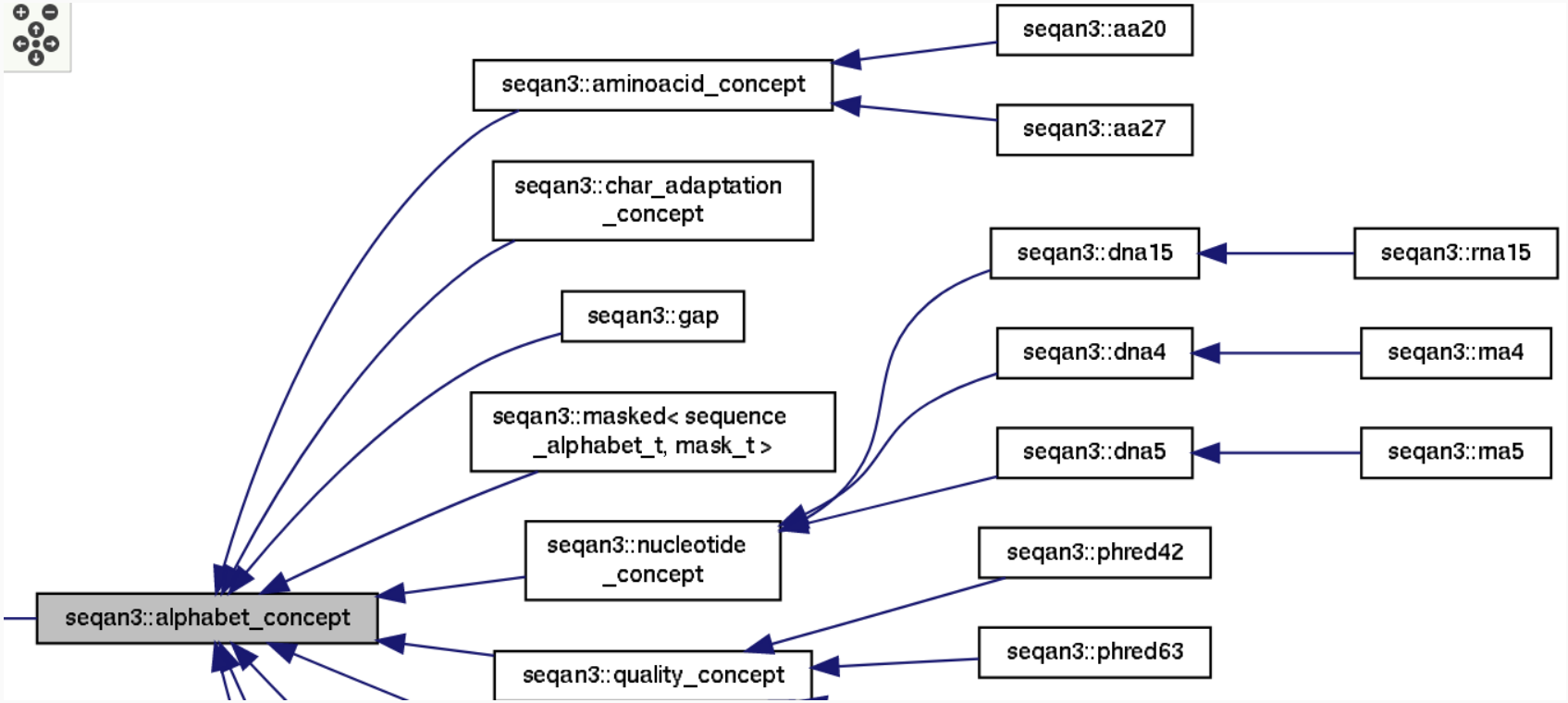
Example:

```
std::vector<int> vec{1, 5, 6, 8, 5};

auto v = vec
       | view::transform([] (int i) { return i*i; })
       | view::drop(2);

std::cout << *v.begin() << '\n'; // prints '36'
```

Introduction to SeqAn3

SeqAn3 Alphabets and views

SeqAn3 – alphabet module



SeqAn3 – alphabet-based views

Reverse complement via view:

```
// note that this _dna5 suffix is required:
std::vector<dna5> vec{"AGATTAC"_dna5};

// just like "GTAATCT"_dna5 but is generated on demand:
auto v = vec | view::reverse | view::complement;

// copy into a new vector to "save" results:
std::vector<dna5> vec2{v};
```

SeqAn3 – alphabet-based views

Translation via view:

```
std::vector<dna5> vec{"ACGTACGTACGTA"_dna5};

// v1 is view of amino acids just like "TYVR"_aa22:
auto v1 = vec | view::translate_single;

// with six-frame translation view is 2-dimensional:
auto v2 = vec | view::translate;
// == [TYVR,RTYV,VRT,YVRT,TYVR,RTY]
```

Accessing only ‘v2[4][1]‘ will result in only exactly three bases being reverse-complemented and translated.

Overview

Introduction to SeqAn3

Small Read Mapper

Outlook

Small Read Mapper

Setup

Setup

Ubuntu >= 18.04

```
sudo apt-get install gcc
```

Ubuntu < 18.04

```
sudo add-apt-repository ppa:ubuntu-toolchain-r/test
sudo apt-get update
sudo apt-get install gcc-7
```

MacOS using Homebrew

```
brew install gcc@7
```

MacOS using MacPorts

```
sudo port install gcc-7
```

Setup

```
mkdir tutorial
cd tutorial
mkdir build

git clone -b 01_setup_app \
    https://github.com/eseiler/SeqAn3_read_mapper.git

git clone --recursive https://github.com/seqan/seqan3.git

cd build
cmake ../SeqAn3_read_mapper -DCMAKE_CXX_COMPILER=g++-7 \
    -DSEQAN3_DIR=../seqan3
make
./mapper
```

Small Read Mapper

Argument Parser

Argument Parser

Aims to provide an easy and feature-rich way to parse command line arguments.

```
./my_program -f -i 3 /tmp/my_file.txt
```

Argument Parser

Aims to provide an easy and feature-rich way to parse command line arguments.

```
./my_program -f -i 3 /tmp/my_file.txt
```

SeqAn 3 Built-in Functionality:

Argument Parser

Aims to provide an easy and feature-rich way to parse command line arguments.

```
./my_program -f -i 3 /tmp/my_file.txt
```

SeqAn 3 Built-in Functionality:

- Deduce options and positional options automatically

Argument Parser

Aims to provide an easy and feature-rich way to parse command line arguments.

```
./my_program -f -i 3 /tmp/my_file.txt
```

SeqAn 3 Built-in Functionality:

- Deduce options and positional options automatically
- Check if the command line call is ill-formed

Argument Parser

Aims to provide an easy and feature-rich way to parse command line arguments.

```
./my_program -f -i 3 /tmp/my_file.txt
```

SeqAn 3 Built-in Functionality:

- Deduce options and positional options automatically
- Check if the command line call is ill-formed
- Provide default values

Argument Parser

Aims to provide an easy and feature-rich way to parse command line arguments.

```
./my_program -f -i 3 /tmp/my_file.txt
```

SeqAn 3 Built-in Functionality:

- Deduce options and positional options automatically
- Check if the command line call is ill-formed
- Provide default values
- Check arguments for contrains (e.g. file extension)

Argument Parser

```
#include <seqan3/argument_parser/all.hpp>

int main(int argc, const char ** argv)
{
    seqan3::argument_parser p("test", argc, argv);

    // specialize all user values you want to parse
    int my_int{0};
    std::string name{};
    bool my_flag{false};

    // for each value add an (positional_)option call
    p.add_option(my_int, 'i', "integer", "desc");
    p.add_positional_option(name, "desc");
    p.add_flag(my_flag, "f", "flag", "desc");

    p.parse(); // in the end, call parse
}
```

Argument Parser

```
try
{
    p.parse(); // in the end, call parse
}
catch (seqan3::parser_invalid_argument const & ext)
{
    // The user did something wrong!
    // Print out a customized error message
    std::cerr << "oops.." << ext.what();
    return -1;
}
catch (seqan3::parser_interruption const &)
{
    // expected behaviour on special requests
    // (e.g. `--help`)
    return 0;
}
```

Argument Parser

Tasks:

1. Read the genome file name via command line.
2. Read in the read sequences file name via command line.
3. Read in the output file name via command line.
4. Add an integer option called ”m/max_error” that will configure the search later on.

Data structures that you will need:

```
argument_parser, std::string
```

Bonus Tasks:

- Restrict the the max error option to positive numbers.
- Set the max error option to required.

Small Read Mapper

Sequence file input

Sequence input

File Format	File Extension
FASTA	`.fa` , `.fasta`
FASTQ	`.fq` , `.fastq`

```
>seq1
CCCCCCCCCCCCCCC
>seq2
CGATCGATC
>seq3
TTTTTTT
```

```
@seq1
CCCCCCCCCCCCCCC
+
IIIIIHIIIIIIIII
@seq2
CGATCGATC
```

Sequence input

Construct a Sequence File Object:

```
sequence_file_input fin{"/tmp/my.fasta"};
```

Can be constructed from:

- a filename (format detection based on extension)
- a stream object (e.g. `std::stringstream`; format has to be specified)

Sequence input

A Sequence File is an **input range over records**.

InputRange Something you can iterate over.

Record Similar to std::tuple.

```
sequence_file_input fin{"/tmp/my.fasta"};

for (auto & rec : fin)
{
    debug_stream << get<field::ID>(rec) << '\n';
    debug_stream << get<field::SEQ>(rec) << '\n';
    //debug_stream << get<field::QUAL>(rec) << '\n';
}
```

Sequence input

A Sequence File is an **input range over records**.

InputRange Something you can iterate over.

Record Similar to std::tuple.

```
sequence_file_input fin{"/tmp/my.fasta"};

for (auto & [ seq, id, qual ] : fin)
{
    debug_stream << id << '\n';
    debug_stream << seq << '\n';
    // debug_stream << qual << '\n';
}
```

Sequence input

Your Task

1. Read in the genome file (FASTA).
2. Read in the reads (FASTQ).
3. Print the first 10 letters of the genome
4. Print the first read's sequence and quality string

Data structures that you will need:

```
sequence_file_input
std::vector<dna5>
debug_stream // for easy printing
view::take   // not mandatory
```

Small Read Mapper

Search Module

Search module

- Indices:
  - FM index, Bi-FM index (based on the SDSL)
  - K-mer index (coming soon)

Search module

- Indices:
  - FM index, Bi-FM index (based on the SDSL)
  - K-mer index (coming soon)
- Search algorithms:
  - one search interface
  - wrapping different search algorithms
  - selecting the best search strategy based on input

Searching an index

```
#include <seqan3/alphabet/nucleotide/dna4.hpp>
#include <seqan3/io/stream/debug_stream.hpp>
#include <seqan3/search/all.hpp>

using namespace seqan3;
using namespace seqan3::literal;

int main()
{
    dna4_vector genome{"AAGCTACGAAACGT"_dna4};
    fm_index index{genome};

    debug_stream << search(index, "ACG"_dna4);

    return 0;
}
```

Search module

- max_error(total{int}, substitution{int},
  insertion{int}, deletion{int})
- max_error_rate(total{double}, substitution{double},
  insertion{double}, deletion{double})
- mode(all / all_best / best)

Piping configs

```
auto cfg =
    search_cfg::max_error(search_cfg::total{1}, ...) |
    search_cfg::mode(search_cfg::best);

search(indexed_genome, "ACG"_dna4, cfg);
```

Search module

- arbitrary order and subset of error types, e.g. max_error(deletion{2})
- mode(all / all_best / best / strata{2})
- output(text_position / index_iterator)
- on_hit(delegate)

Search Module

Tasks:

1. Build an index on the reference genome.
2. Search the reads with max_error.
3. Output the hit positions in the text.

Data structures and functions that you will need:

```
fm_index search(index, queries, cfg);
max_error(total{0}, substitution{0},
          insertion{0}, deletion{0});
mode(all / best);
```

Bonus Tasks:

- Introduce a flag to toggle between ”best” and ”all”.
- Introduce a flag to toggle between Hamming distance (substitutions only) and Edit distance (all error types).

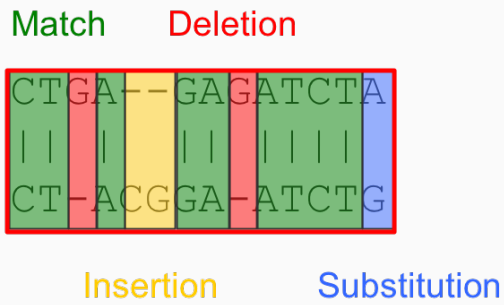
Small Read Mapper

Alignment

Alignment

Compute sequence similarity between pair of sequences (dna, amino-acid)

- global/semi-global
- local




```
EMBOSS_001      1 -TATGGAGAGAATAAAAGAACTGA--GAGATCTAATGTCGCAGTCCCGCA     47
EMBOSS_001      1 ATATGGAAAGAATAAAAGAACT-ACGGA-ATCTGATGTCGCAGTCTCGCA     48

EMBOSS_001     48 CTCGCGAGATACTCACTAAGACCACTGTGGACCATATGGCCATAATCAAA     97
EMBOSS_001     49 CTCGCGAGATACTAACAAAAACCACAGTGGACCATATGGCCATAATTAAG     98

EMBOSS_001     98 AAGTACACATCAGGAAGGCAAGAGAAGAACCCCG-CGCTCAGAATGAAGT    146
EMBOSS_001     99 AAGTACACATCAGGGAGACAGGAAAAGAA-CCCGTCACTTAGGATGAAAT    147

EMBOSS_001    147 GGATGATGGCAATGAGATACCCAATTACAGCAGACAAGAGAATAATGGAC    196
EMBOSS_001    148 GGATGATGGCAATGAAATATCCAATTACAGCTGACAAGAGGATAACAGAA    197

EMBOSS_001    197 ATGATTCCAGAGAGGAATGAACAAGGACAAACCCTCTGGAGCAAAACAAA    246
EMBOSS_001    198 ATGGTTCCTGAGAGAAATGAGCAAGGACAAACCCTATGGAGTAAAATGAG    247

EMBOSS_001    247 CGATGCTGGATCAGACCGAGTGATGGTATCACCTCTGGCCGTAACATGGT    296
EMBOSS_001    248 TGATGCCGGGTCAGATCGAGTGATGGTATCACCTTTGGCGGTGACATGGT    297

EMBOSS_001    297 GGAATAGGAATGGCCCAACAACAAGTACAGTTCATTACCCTAAGGTATAT    346
EMBOSS_001    298 GGAATAGAAATGGACCAGTGACAAGTACGGTTCATTATCCAAAAGTCTAC    347

EMBOSS_001    347 AAAACTTATTTCGA-AAAGGTCGAAAGGTTGAAACATGGTACCTTCGGCC    395
EMBOSS_001    348 AAGACTTATTTTGATAAA-GTCGAAAGGTTAAAACATGGAACCTTTGGCC    396

EMBOSS_001    396 CTGTCCACTTCAGAAATCAAGTTAAAATAAG--GAGGAGAGTTGATACAA    443
EMBOSS_001    397 CTGTCCATTTTAGAAACCAAGTCAAAATACGCCGA--AGAGTTGACATAA    444
```

Alignment

- SeqAn implementation allows for over 500 variations of standard alignment kernel
- SeqAn3 offers generic and easy-to-use interface to handle configurations and to check for consistency at compile time or runtime.
- Let's first have a look at an example:

Alignment

Alignment Configurations:

- Dedicated namespace `align_cfg`
- Configurations for different settings: `score`, `gap`, `global`, `sequence_ends`, `output`
  - Variable configurations: `gap(gap_cost{-1})`
  - Static configurations: `output<align_result_key::score>`
  - Hybrid configurations: `sequence_ends<free_ends_at::seq1>` | `sequence_ends<>(free_ends_at::seq1)`
- Configuration shortcuts: `edit <= global | score(nucleotide_scoring_scheme{}) | gap(gap_cost{-1})`

Alignment

```
#include <seqan3/alignment/all.hpp>
#include <seqan3/alphabet/nucleotide/dna4.hpp>
#include <seqan3/io/stream/debug_stream.hpp>

using namespace seqan3;
using namespace seqan3::literal;
```

Alignment

```
int main()
{
    dna4_vector seq1{"ACGTGATG"_dna4};
    dna4_vector seq2{"AGTGATACT"_dna4};

    auto cfg = align_cfg::global                          |
        align_cfg::score(nucleotide_scoring_scheme{
            match_score{4}, mismatch_score{-5}}           |
        align_cfg::gap(gap_score{-1},
            gap_open_score{-11})                          |
        align_cfg::output<align_result_key::trace>;
    // ...
}
```

Alignment

```
int main()
{
    // ...
    auto rng = align_pairwise(std::tie(seq1, seq2),
                              cfg));
    for (auto && res : rng)
    {
        auto && [gap1, gap2] = res.trace();
        debug_stream << res.score() <<
             gap1 << '\n' <<
             gap2 << '\n';
    }
    return 0;
}
```

Alignment

Tasks:

1. Get the respective slice of the genome
2. Compute the respective alignments with edit distance.
3. Print the alignment.

Small Read Mapper

Alignment File Output

Alignment File Output

The Alignment File structure is the same as for Sequence File.

Key features:

- The file object is treated as a range of records.
- The format is automatically deduced by the filename.
- You can select the fields (columns) to read/write.

SAM format

```
@HD VN:1.6 SO:coordinate
@SQ SN:ref LN:47
ref    516 ref  1  0 14M2D31M   *  0   0 AGCATGTTAGATAAGATAGCTGTGCTAGTAGGCAGTCAGCGCCAT *
r001    99 ref  7 30 14M1D3M    = 39  41 TTAGATAAAGGATACTG *
*      768 ref  8 30 1M         *  0   0 *                 * CT:Z:.;Warning;Note=Ref wrong?
r002     0 ref  9 30 3S6M1D5M   *  0   0 AAAAGATAAGGATA    * PT:Z:1;4;+;homopolymer
r003     0 ref  9 30 5H6M       *  0   0 AGCTAA            * NM:i:1
r004     0 ref 18 30 6M14N5M    *  0   0 ATAGCTTCAGC       *
r003  2064 ref 31 30 6H5M       *  0   0 TAGGC             * NM:i:0
r001   147 ref 39 30 9M         =  7 -41 CAGCGGCAT         * NM:i:1
```

Col	Field	Type	Regexp/Range	Brief description
1	QNAME	String	`[!-?A-~]{1,254}`	Query template NAME
2	FLAG	Int	$[0,2^{16}-1]$	bitwise FLAG
3	RNAME	String	`\*|[!-()+-<>-~][!-~]*`	Reference sequence NAME
4	POS	Int	$[0,2^{31}-1]$	1-based leftmost mapping POSition
5	MAPQ	Int	$[0,2^{8}-1]$	MAPping Quality
6	CIGAR	String	`\*|([0-9]+[MIDNSHPX=])+`	CIGAR string
7	RNEXT	String	`\*|=|[!-()+-<>-~][!-~]*`	Ref. name of the mate/next read
8	PNEXT	Int	$[0,2^{31}-1]$	Position of the mate/next read
9	TLEN	Int	$[-2^{31}+1,2^{31}-1]$	observed Template LENgth
10	SEQ	String	`\*|[A-Za-z=.]+`	segment SEQuence
11	QUAL	String	`[!-~]+`	ASCII of Phred-scaled base QUALity+33

Alignment File Output

Construct the alignment file

```
alignment_file_output fout{"/tmp/my.sam"};
```

You may specialize the following:

- Fields: The fields to write.
- Valid formats: Which formats does yours file accept.
- Stream: Which stream type you want to read from/print to.

Alignment File Output

Again the Alignment File is a **range of records**.

So the easiest way is to push back a record to the file.

Alignment File Output

Again the Alignment File is a **range of records**.

So the easiest way is to push back a record to the file. The

SeqAn3 Record:

```
record<type_list, field_list>
```

where the type_list specifies the types that will give the record, and field_list gives a list of fields corresponding to the types.

Alignment File Output

Again the Alignment File is a **range of records**.

So the easiest way is to push back a record to the file. The

SeqAn3 Record:

```
record<type_list, field_list>
```

where the type_list specifies the types that will give the record, and field_list gives a list of fields corresponding to the types.

```
using types_t  = type_list<dna5_vector, std::string>;
using fields_t = fields<field::SEQ, field::ID>;

using record_t = record<types_t, fields_t>;
```

Alignment File Output

The header object is stored in the file object and written once before the first record is written.

```
alignment_file_output fout{"/tmp/my.sam"};

// always give the reference information [required]
// The following adds information for the reference
// "ref" of length 1000 with no extra information.

fout.header().ref_dict["ref"] = {1000, std::string{}};
```

Alignment File Output

```
#include <seqan3/io/alignment_file/all.hpp>

using namespace seqan3;
using namespace seqan3::literal;

int main()
{
    using types_t  = type_list<dna5_vector, std::string>;
    using fields_t = fields<field::SEQ, field::REF_ID>;
    using record_t = record<types_t, fields_t>;

    dna5_vector seq = "ACGCGATCG"_dna5;
    std::string id = "ref";

    alignment_file_output fout{"/tmp/my.sam"};
    fout.header().ref_dict[id] = {1000, std::string{}};

    fout.push_back(record_t{seq, id});
}
```

Overview

Introduction to SeqAn3

Small Read Mapper

Outlook

Expect a first release around Christmas!

Follow and/or star us on GitHub:
https://github.com/seqan/seqan3

Subscribe to our mailing list:
https://lists.fu-berlin.de/listinfo/seqan-dev

Follow us on Twitter: @SeqAnLib