



# SeqAn - UGM 2016

## Parsing Command Line Arguments

**Svenja Mehringer**

Algorithmische Bioinformatik

FU Berlin, Germany

# The SeqAn Command Line Parser (CLP)

Why use the SeqAn CLP ?

# The SeqAn Command Line Parser (CLP)

Why use the SeqAn CLP ?

- **Robust parsing** of command line arguments.

# The SeqAn Command Line Parser (CLP)

## Why use the SeqAn CLP ?

- **Robust parsing** of command line arguments.
- **Simple verification** of arguments.

# The SeqAn Command Line Parser (CLP)

## Why use the SeqAn CLP ?

- **Robust parsing** of command line arguments.
- **Simple verification** of arguments.
- **Automatically** generated and nicely formatted **help screens** when called with `--help`. You can also export this help to **HTML** and **man pages**.

# The SeqAn Command Line Parser (CLP)

## Why use the SeqAn CLP ?

- **Robust parsing** of command line arguments.
- **Simple verification** of arguments.
- **Automatically** generated and nicely formatted **help screens** when called with `--help`. You can also export this help to **HTML and man pages**.
- The CLP helps you to generate **nodes for workflow engines** such as KNIME or Galaxy.

# A First Working Example

„modify\_string“

```
# modify_string --uppercase -i 2 "This is some text!"  
ThIs iS SoMe TeXt!  
# modify_string "This is some text!" --lowercase -i 1  
this is some text!
```

# A First Working Example

Including CLP with `#include <seqan/arg_parse.h>`

```
#include <iostream>

#include <seqan/arg_parse.h>

int main(int argc, char const ** argv)
{
```

Create an ArgumentParser object `parser`  
with the project or app name „modified string“.

```
// Setup ArgumentParser.
seqan::ArgumentParser parser("modify_string");
```



# A First Working Example

Now you can add arguments and options:

- 1) `addArgument(parser, ArgParseArgument)` identified by position
- 2) `addOption(parser, ArgParseOption)` identified by name

# A First Working Example

Now you can add arguments and options:

1) `addArgument(parser, ArgParseArgument)` identified by position

2) `addOption(parser, ArgParseOption)` identified by name

→ `ArgParseArgument(argumentType [,argumentLabel  
[,isListArgument [,numberOfArgument]]]);`

```
addArgument(parser, seqan::ArgParseArgument(  
    seqan::ArgParseArgument::STRING, "TEXT"));
```

# A First Working Example

Now you can add arguments and options:

1) `addArgument(parser, ArgParseArgument)` identified by position

2) `addOption(parser, ArgParseOption)` identified by name

→ `ArgParseOption(shortName, longName, helpText, argumentType  
[, argumentLabel[, isList[, numValues]]]);`

```
addOption(parser, seqan::ArgParseOption(  
    "i", "period", "Period to use for the index.",  
    seqan::ArgParseArgument::INTEGER, "INT"));  
addOption(parser, seqan::ArgParseOption(  
    "U", "uppercase", "Select to-uppercase as operation."));
```

# A First Working Example

Next we parse the command line using the seqan function `parse`

```
// Parse command line.  
seqan::ArgumentParser::ParseResult res = seqan::parse(parser, argc, argv);
```

# A First Working Example

Next we parse the command line using the seqan function `parse`

```
// Parse command line.  
seqan::ArgumentParser::ParseResult res = seqan::parse(parser, argc, argv);
```

Different types for the result:

1. `seqan::ArgumentParser::PARSE_ERROR`
2. `seqan::ArgumentParser::PARSE_OK`
3. neither in case of special behaviour (e.g. `--help`)

```
if (res != seqan::ArgumentParser::PARSE_OK)  
    return res == seqan::ArgumentParser::PARSE_ERROR;
```

# A First Working Example

Finally, we access the values using the Argument parser.

We will use the following functions:

- `getArgumentValue(destination, parser, pos[, no])`
- `getOptionValue(destination, parser, name[, pos])`
- Special case example: `isSet(parser, name)`

functions can be found in the API documentation

<http://docs.seqan.de/seqan/master/?p=ArgumentParser>

# A First Working Example

```
// Extract option values and print them.
unsigned period = 0;
getOptionValue(period, parser, "period");
bool toUppercase = isSet(parser, "uppercase");
seqan::CharString text;
getArgumentValue(text, parser, 0);

std::cout << "period \t" << period << '\n'
           << "uppercase\t" << toUppercase << '\n'
           << "text \t" << text << '\n';
```

<http://seqan.readthedocs.org>



## Assignment 1:

Copy A First working Example and try out the commads:

```
# modify_string --uppercase -i 2 "This is some text!"
# modify_string "This is some text!" --lowercase -i 1
```

# A First Working Example



## Assignment 2:

Adjust the program from above to also accept an option to convert characters to lower case, just as it accepts options to convert characters to upper case. The long name should be `--lowercase`, the short name should be `-L`. As for the `--uppercase` option, the program should print whether the flag was set or not.

## Hint

Copy the two lines for defining the `-uppercase` option and replace the strings appropriately.



# Using Default Values

You can set a default value to an ArgumentParser option with

```
SetDefaultValue(option, value)
```

Note: value has to be a string (or anything convertible to std::stringstream)



## Assignment 3:

Adjust the previous program to accept the default value „1“ for the period by adding the setDefaultValue() function.

# Best Practice

1. Store all arguments and options in a struct
2. Extract the the argument parsing into its own function

```
struct ModifyStringOptions
{
    unsigned period;
    bool toUppercase;
    bool toLowercase;
    seqan::CharString text;

    ModifyStringOptions() :
        period(1), toUppercase(false), toLowercase(false)
    {}
};
```

# Setting Restrictions

We can:

- Set maximum and minimum values

```
setMinValue(parser, "i", "10");  
setMaxValue(parser, "integer-value", "20");
```

- Mark options as being required.

```
setRequired(parser, "i");
```

- Set lists of valid values for each option.

```
setValidValues(parser, "distance-model", "HAMMING EDIT");
```

# More Option and Argument Types

## 1. Input or Output file names:

- `ArgParseArgument::INPUT_FILE`
- `ArgParseArgument::OUTPUT_FILE`

Note: You can use restrictions for checking file name extensions

```
setValidValues(parser, "input-file", "txt");  
setValidValues(parser, "output-file", "txt");
```



### Assignment 5:

Replace the argument TEXT by a command line option `-I/--input-file` in the program above. The program should then read in the text instead of using the command line argument.

# More Option and Argument Types

## 2. Tuples

Define an option or argument to have a fixed number of values.

```
addOption(parser, seqan::ArgParseOption(  
    "r", "range", "The range to modify.",  
    seqan::ArgParseArgument::INTEGER, "BEGIN END",  
    false, 2));
```

Use `getOptionValue` with an extra parameter to access the values

```
unsigned rangeBegin = 0, rangeEnd = 0;  
getOptionValue(rangeBegin, parser, "range", 0);  
getOptionValue(rangeEnd, parser, "range", 1);
```

# Embed Rich Documentation

You can set the short description, the version string, date,

```
setShortDescription(parser, "String Modifier");  
setVersion(parser, "1.0");  
setDate(parser, "July 2012");
```

and a synopsis.

```
addUsageLine(parser,  
              "[\\fIOPTIONS\\fP] \\\"\\fITEXT\\fP\\\"");  
addDescription(parser,  
               "This program allows simple character modifications to "  
               "each i-th character.");
```

# Embed Rich Documentation

Structure your options into sections using addSection:

```
// initialize parser and add information  
addSection(parser, „First Section“)  
  
// add some options  
addSection(parser, „Second Section  
// add some more options
```

# Embed Rich Documentation

Add some helpful examples to your help page:

```
addTextSection(parser, "Examples");

addListItem(parser,
    "\\fBmodify_string\\fP \\fB-U\\fP \\fIveryverylongword\\fP",
    "Print upper case version of \"veryverylongword\"");
addListItem(parser,
    "\\fBmodify_string\\fP \\fB-L\\fP \\fB-i\\fP \\fI3\\fP \\fIveryvery",
    "Print \"veryverylongword\" with every third character "
    "converted to upper case.");
```



## Last Assignment:

Copy the complete program and take a look at the resulting help page.

Export the help-page to html or man and take a look at them.

```
# modify_string --export-help html > modify_string.html
```

```
# modify_string --export-help man > modify_string.man
```