

Flags-Template

name: csl

E-Mail: 3079625093@qq.com

```
1  _|_| |_|
2  _| |_| |_|_| |_|_| |_|_|
3  _|_|_| |_| |_| |_| |_|_|
4  _| |_| |_| |_| |_| |_|_|
5  _| |_| |_|_| |_|_| |_|_| |_|
6  _|_|
7  |_|_|
8
9  _| |_|_|_| |_|_| |_|_| |_|_| |_|_| |_|_| |_|_|
10 _|_|_| |_|_| |_|_| |_|_| |_|_| |_|_| |_|_| |_|_| |_|_|
11 _| |_|_|_| |_| |_| |_| |_| |_| |_| |_| |_| |_| |_|_|_|
12 _| |_| |_|_| |_| |_| |_| |_| |_| |_| |_| |_| |_| |_|
13 |_|_| |_|_|_| |_| |_| |_| |_|_| |_| |_|_| |_|_| |_|_|
14 |_|
15 |_|
```

Overview

this is a simple 'program-command-line-parameter-parsing' library using cpp-template.

the main functions:

- Add command line parameters to the specified program and set the relevant properties of the command line parameters;
- Parse the passed in parameters based on the set command line parameters;
- During parsing, identify and check the command line parameters (such as wrong type, wrong option name, inconsistent selectability);

Usage

Example for Source Code

```
1  #include "flags.hpp"
2
3  using namespace ns_flags;
4
5  int main(int argc, char const* argv[]) {
6      /**
7       * @brief try-catch is not necessary but it is strongly recommended,
8       * because you can get a lot of advice when there are errors in your code
9       */
10     try {
```

```

11 ns_flags::ArgParser parser;
12 /**
13  * @brief define some kinds of arguments
14  * [int, std::string, bool, double]
15  * std::vector<int, std::string, bool, double>
16  */
17 parser.add_arg<ArgType::INT>("id", 0, "the id of current thread");
18 parser.add_arg<ArgType::STRING>("usr", "null", "the name of usr");
19 parser.add_arg<ArgType::BOOL>("sex", true,
20     "the sex of usr [male: true, female: false]");
21 parser.add_arg<ArgType::DOUBLE>("height", 1.7, "the height of usr",
22     OptProp::REQUIRED);
23 parser.add_arg<ArgType::INT_VEC>("ids", {1, 2, 3}, "the ids of threads");
24 parser.add_arg<ArgType::STRING_VEC>("langs", {"cpp", "python"},
25     "the used languages of usr");
26 parser.add_arg<ArgType::BOOL_VEC>("choice", {true, false},
27     "the choice of usr");
28 parser.add_arg<ArgType::DOUBLE_VEC>("scores", {2.3, 4.5},
29     "the score of usr");
30 /**
31  * @brief set version and help docs
32  * @attention if you do not set the help docs, then the help docs
33  * will generate automatically
34  */
35 parser.set_version("2.0");
36 // parser.set_help("");
37
38 parser.set_nopt_arg<ArgType::STRING_VEC>({""}, OptProp::REQUIRED);
39 /**
40  * @brief finally, you can set up the parser and then use these arguments
41  */
42 parser.setup_parser(argc, argv);
43
44 /**
45  * @brief print the info of arguments
46  */
47 std::cout << parser.get_nopt_argi() << std::endl;
48 for (const auto& [key, value] : parser.get_args())
49     std::cout << value << std::endl;
50
51 /**
52  * @brief use the arguments
53  */
54 auto id = parser.get_argv<ArgType::INT>("id");
55 std::cout << "the 'id' I get is: " << id << std::endl;
56 } catch (const std::exception& e) {
57     std::cerr << e.what() << '\n';
58 }
59 return 0;
60 }

```

Output

if you want to over view the example command lines and outputs, please click [the log file](#).

if run command line:

```
1 | ./flags hello "I'm" flags!
```

will output:

```
1 | [ error from lib-flags ] the property of the '--height' is 'OptProp::required', but you didn't pass the
argument(s)
```

if run command line:

```
1 | /flags hello "I'm" flags! --height 98.8 --sex true --usr csl --id 12 --choice true false true --ids 12
34 123 --scores 12.3 45.6 78.9 --langs cpp java python html
```

will output:

```
1 | {'name': nopt_arg, 'value': [hello, I'm, flags!], 'default': [], 'desc': arguement(s) without any
option}
2 | {'name': choice, 'value': [true, false, true], 'default': [true, false], 'desc': the choice of usr}
3 | {'name': ids, 'value': [12, 34, 123], 'default': [1, 2, 3], 'desc': the ids of threads}
4 | {'name': scores, 'value': [12.3, 45.6, 78.9], 'default': [2.3, 4.5], 'desc': the score of usr}
5 | {'name': langs, 'value': [cpp, java, python, html], 'default': [cpp, python], 'desc': the used
langusges of usr}
6 | {'name': height, 'value': 98.800000, 'default': 1.700000, 'desc': the height of usr}
7 | {'name': sex, 'value': true, 'default': true, 'desc': the sex of usr [male: true, female: false]}
8 | {'name': usr, 'value': csl, 'default': null, 'desc': the name of usr}
9 | {'name': id, 'value': 12, 'default': 0, 'desc': the id of current thread}
10 | {'name': help, 'value': false, 'default': false, 'desc': get help docs of this program}
11 | {'name': version, 'value': , 'default': 1.0, 'desc': the version of this program}
12 | the 'id' I get is: 12
```

if run command line:

```
1 | ./flags --help
```

will output:

```
1 | Usage: ./flags [nopt-arg(s)] [--option target(s)] ...
2 |
3 |      Options      property      Default Value      Describes
4 | -----
5 | --nopt-arg(s)    required      []                arguement(s) without any option
6 |
7 | --choice         optional      [true, false]     the choice of usr
8 | --ids            optional      [1, 2, 3]         the ids of threads
9 | --scores         optional      [2.3, 4.5]        the score of usr
10 | --langs          optional      [cpp, python]     the used langusges of usr
11 | --height         required      1.700000          the height of usr
12 | --sex            optional      true              the sex of usr [male: true, female: false]
13 | --usr            optional      null              the name of usr
```

```

14  --id          optional    0          the id of current thread
15
16  --help        optional    help docs   get the help docs of this program
17  --version     optional    0.0.1      get the version of this program
18
19  program help docs

```

if run command line:

```

1  | ./flags --version

```

will output:

```

1  | ./flags version: 2.0

```

if run command line:

```

1  | ./flags --nema 12

```

will output:

```

1  some error(s) happened in the command line:
2  [ error from lib-flags ] the option named '--nema' is invalid, use '--help' option for help.

```

Apis

Argument Types

Here are the types you can use in the 'argument-parser':

```

1  using INT = int;
2  using DOUBLE = double;
3  using BOOL = bool;
4  using STRING = std::string;
5  using INT_VEC = std::vector<int>;
6  using DOUBLE_VEC = std::vector<double>;
7  using BOOL_VEC = std::vector<bool>;
8  using STRING_VEC = std::vector<std::string>;

```

Option Property

```

1  enum class OptProp {
2      /**
3       * @brief options
4       */
5      OPTIONAL,
6      REQUIRED
7  };

```

Arguement Info

These members are config objects in an 'arguement-info' object:

```
1  std::string _name;
2
3  OptProp _prop;
4
5  std::any _value;
6  std::any _default_value;
7
8  std::string _desc;
```

Apis in the ArgParser

ArgParser()

```
1  /**
2   * @brief the default and only constructor for ArgParser
3   */
```

```
template void add_arg(const std::string &name, const Type &default_value, const std::string &desc,
OptProp prop = OptProp::OPTIONAL)
```

```
1  /**
2   * @brief add a arguement to the parser
3   *
4   * @tparam Type the type of the arguement
5   * @param name the name of the arguement
6   * @param default_value the default value of the arguement
7   * @param desc the describe of the arguement
8   * @param prop the property of this option
9   */
```

```
template void set_nopt_arg(const Type &default_value, OptProp prop = OptProp::OPTIONAL, const
std::string &desc = "arguement(s) without any option")
```

```
1  /**
2   * @brief Set the nopt-arg(s) arguement
3   *
4   * @tparam Type the type of arguement
5   * @param default_value the default value of the nopt-arg(s) arguement
6   * @param prop the property of this option
7   * @param desc
8   */
```

```
template inline const Type &get_nopt_argv() const
```

```

1  /**
2   * @brief Get the no-option argument's value
3   *
4   * @tparam Type the vaule type
5   * @return const Type&
6   */

```

inline const ArgInfo get_nopt_argi() const

```

1  /**
2   * @brief Get the no-option argument info object
3   *
4   * @return const ArgInfo
5   */

```

inline std::size_t get_argc() const

```

1  /**
2   * @brief get the count of the arguments in the parser
3   *
4   * @return size_t
5   */

```

inline const ArgInfo &get_argi(const std::string &name) const

```

1  /**
2   * @brief Get the arg info object in the parser according to the name
3   *
4   * @param name the name of the argument
5   * @return const ArgInfo&
6   */

```

inline const auto &get_args() const

```

1  /**
2   * @brief Get the all arguments in the parser
3   *
4   * @return const auto&
5   */

```

template inline const Type &get_argv(const std::string &name) const

```

1  /**
2   * @brief Get the value of an argument according to name
3   *
4   * @tparam Type the type of this argument
5   * @param name the name of this argument
6   * @return Type&
7   */

```

template inline const Type &get_argdv(const std::string &name) const

```
1  /**
2   * @brief Get the default value of an argument according to name
3   *
4   * @tparam Type the type of this argument
5   * @param name the name of this argument
6   * @return const Type&
7   */
```

inline const std::string &get_argdc(const std::string &name) const

```
1  /**
2   * @brief Get the describe of the argument named 'name'
3   *
4   * @param name
5   * @return const std::string&
6   */
```

void setup_parser(int argc, char const *argv[])

```
1  /**
2   * @brief Set the up the parser
3   *
4   * @param argc the count of the arguments
5   * @param argv the values of the arguments
6   */
```

inline void set_help(const std::string &str)

```
1  /**
2   * @brief Set the help docs string for the parser
3   *
4   * @param str the help str to set
5   */
```

inline void set_version(const std::string &str)

```
1  /**
2   * @brief Set the version of the program
3   *
4   * @param str the version str
5   */
```