

main.cxx

```
1 #include <cxopts.hpp> // command line argument parser library
   // written by jarro2783 (https://github.com/jarro2783/cxopts)
2 #include <ui.hxx>
3
4 auto main(int argc, char** argv) -> int{
5     // flags for program arguments
6     bool upAlphaFlag = false;
7     bool lowAlphaFlag = false;
8     bool numFlag = false;
9     bool specialCharFlag = false;
10    bool guiFlag = false;
11
12    // other variable change by the flags
13    unsigned int length = 0;
14    int state = -1; // execution state (-1 for init value)
15    bool flagEnabled = false;
16
17    // specifies options
18    cxopts::Options options("APCSP Create Task", "Random Password
    Generator");
19    options.add_options()
20        ("A,upper", "Include upper case alphabets")
21        ("a,lower", "Include lower case alphabets")
22        ("n,number", "Include numbers")
23        ("s,special", "Include special characters")
24        ("g,gui", "Run the GUI version of program, other passed
    arguments are ignored")
25        ("h,help", "Print help")
26        ("l,length", "Set the length of password", cxopts::value<
    int>());
27
28
29    // parse options from argv with fail safe
30    cxopts::ParseResult result;
31    try {
32        result = options.parse(argc, argv);
33    } catch (const cxopts::exceptions::parsing &e) {
34        printHelp();
35        std::cerr << e.what() << "\n"; // show error
36        return FAIL;
37    }
38
39    // sets flags and option value from options(argc and argv)
40    // help flag (show help doc and exit)
41    if (result.count("help")) {
42        printHelp();
43        return SUCCESS;
44    }
45
46    // GUI flag
47    if (result.count("gui")) guiFlag = true;
48
49    // char flags
50    if (result.count("upper")) upAlphaFlag = true;
51    if (result.count("lower")) lowAlphaFlag = true;
```

```

52     if (result.count("number")) numFlag = true;
53     if (result.count("special")) specialCharFlag = true;
54     flagEnabled = checkFlags(upAlphaFlag, lowAlphaFlag, numFlag,
55                               specialCharFlag);
56
57     // length option
58     if (result.count("length") && !guiFlag && flagEnabled) {
59         length = result["length"].as<int>();
60     } else if (!guiFlag && flagEnabled) { // print out error when
61         // no character flag is enabled but only length option specified
62         printHelp();
63         printLine("Error: no length argument");
64         return FAIL;
65     }
66
67     // run GUI version if -g option is present
68     if (guiFlag) {
69         state = rungui();
70     } else if (length > 0 && flagEnabled) { // otherwise run CUI
71         // version
72         state = runcui(length, upAlphaFlag, lowAlphaFlag, numFlag,
73                        specialCharFlag);
74     } else { // if no argument specified, report error and set
75         // execution state 1 (failure)
76         printHelp();
77         printLine("Error: No argument specified");
78         state = FAIL;
79     }
80     return state;
81 }

```

passgen.hxx

```

1  #ifndef PASSGEN_HXX
2  #define PASSGEN_HXX
3
4  #include <cstdlib>
5  #include <ctime>
6  #include <exception>
7
8  namespace PassGen {
9      /**
10       * @brief Get the list of upper alphabet letters
11       * @return char* - List of upper alphabet letters
12       */
13       auto getUpperAlpha() -> char*;
14
15       /**
16       * @brief Get the list of lower alphabet letters
17       * @return char* - List of lower alphabet letters
18       */
19       auto getLowerAlpha() -> char*;
20
21       /**
22       * @brief Get the list of numbers
23       * @return char* - List of numbers
24       */

```

```

25     auto getNumber() -> char*;
26
27     /**
28     * @brief Get the list of special characters
29     * @return char* - List of special characters
30     */
31     auto getSpecialChars() -> char*;
32
33     /**
34     * @brief generate the password from list of characters and
35     * specified length
36     * @param charList list of characters (pure C string)
37     * @param len length of password to be generated (int)
38     * @return (char*) - generated password
39     */
40     auto passGen(const char *charList, const unsigned int& len) ->
41     char*;
42
43     // exceptions for the functions in PassGen namespace
44     namespace exceptions {
45         // base exception class
46         class exception : public std::exception {
47             public:
48                 // constructor (sets m_message from msg argument)
49                 explicit exception(char* msg) {m_message = msg;};
50                 /**
51                 * @brief show the content of the error message
52                 * @return (char*) pure C string of error message
53                 */
54                 auto what() -> char* {return m_message;};
55             private:
56                 char* m_message = nullptr; // error message (init
57                 with nullptr)
58             };
59
60             // exception to be thrown when something is failed to
61             allocate its memory
62             class memoryAllocationFailiure : public exception {
63             public:
64                 explicit memoryAllocationFailiure() : exception((
65                 char*)"Couldn't allocate memory space!") {};
66                 // call construct from the base class supplied with
67                 specified error message
68             };
69
70             // exception to be thrown when there is no content in
71             charList (PassGen::passgen())
72             class noCharList : public exception {
73             public:
74                 explicit noCharList() : exception((char*)"No
75                 character list specified!") {};
76                 // call construct from the base class supplied with
77                 specified error message
78             };
79
80             // exception to be used when unknown error occurred
81             class unknownError : public exception {

```

```

73         public:
74             explicit unknownError() : exception((char*)"Unknown
Error Caught : Mark me 0 :(") {};
```

```

75             // call construct from the base class supplied with
specified error message
76         };
77     } // PassGen::exceptions
78 } // PassGen
79
80 #endif // PASSGEN_HXX

```

passgen.cxx

```

1  #include <utils.hxx>
2  #include <passgen.hxx>
3
4  auto PassGen::getLowerAlpha() -> char* {
5      const int numOfLetters = 26;
6      char* output = nullptr; // initialize pointer
7      output = (char*)malloc((numOfLetters + 1) * sizeof(char)); //
allocate memory for 26 letters and a terminate character
8      if (output == nullptr) {throw PassGen::exceptions::
memoryAllocationFailiure(); return nullptr;} // check if memory
allocation is failed
9      const int offset = 97; // 97th letter in ASCII (a)
10     // adds 26 letters (a-z)
11     for (int i = 0; i < numOfLetters; i++) {
12         output[i] = (char)(offset + i);
13     }
14     output[numOfLetters] = '\0'; // add terminate character at end
15     return output;
16 }
17
18 auto PassGen::getUpperAlpha() -> char* {
19     const int numOfLetters = 26;
20     char* output = nullptr; // initialize pointer
21     output = (char*)malloc((numOfLetters + 1) * sizeof(char)); //
allocate memory for 26 letters + terminate character
22     if (output == nullptr) {throw PassGen::exceptions::
memoryAllocationFailiure(); return nullptr;} // check if memory
allocation is failed
23     const int offset = 65; // 65th letter in ASCII (A)
24     // adds 26 letters (A-Z)
25     for (int i = 0; i < numOfLetters; i++) {
26         output[i] = (char)(offset + i);
27     }
28     output[numOfLetters] = '\0'; // add terminate character at end
29     return output;
30 }
31
32 auto PassGen::getNumber() -> char* {
33     const int numOfLetters = 10;
34     char* output = nullptr; // initialize pointer
35     output = (char*)malloc((numOfLetters + 1) * sizeof(char));
36     if (output == nullptr) {throw PassGen::exceptions::
memoryAllocationFailiure(); return nullptr;} // check if memory
allocation is failed

```

```

37     const int offset = 48; // 48th letter in ASCII (0)
38     // adds 10 letters (0-9)
39     for (int i = 0; i < numOfLetters; i++) {
40         output[i] = (char)(offset + i);
41     }
42     output[numOfLetters] = '\0'; // add terminate character at end
43     return output;
44 }
45 }
46
47 auto PassGen::getSpecialChars() -> char* {
48     const int numOfLetters = 31; // 31 symbols
49     char* output = nullptr; // initialize pointer
50     output = (char*)malloc((numOfLetters + 1) * sizeof(char));
51     if (output == nullptr) {throw PassGen::exceptions::
memoryAllocationFailiure(); return nullptr;} // check if memory
allocation is failed
52
53     int ind = 0; // index in the output list
54
55     // loop config and range exclusion config
56     const int start = 33; // loop through ASCII #33
57     const int end = 127; // to #127
58
59     const int numStart = 48; // ASCII range that represents number
(#48
60     const int numEnd = 57; // to #57)
61
62     const int upperStart = 65; // ASCII range that represents upper
case alphabets (#65
63     const int upperEnd = 90; // to #90)
64
65     const int lowerStart = 97; // ASCII range that represents lower
case alphabets (#97
66     const int lowerEnd = 122; // to #122)
67
68     // adds special characters to output
69     for (int i = start; i < end; i++) {
70         if ((numStart <= i && i <= numEnd) || (upperStart <= i && i
<= upperEnd) || (lowerStart <= i && i <= lowerEnd)) {
71             continue; // skip at the number, upper case and lower
case alphabets
72         }
73         output[ind] = (char)(i);
74         ind++;
75     }
76     output[numOfLetters] = '\0'; // add terminate character at end
77     return output;
78 }
79
80 auto PassGen::passGen(const char *charList, const unsigned int& len
) -> char* {
81     if (strSize(charList) == 0) {throw PassGen::exceptions::
noCharList(); return nullptr;}
82
83     std::srand(time(nullptr)); // set random seed to current time
84

```

```

85     unsigned int randomCharPos = 0; // position of charList which
      will be randomly selected
86     const char termChar = '\0';
87     const char backSlash = '\\';
88     const int charListSize = strSize(charList);
89     char currentLetter = 0;
90     char previousLetter = 0;
91
92     char* output = new char[len+1]; // length of password + 1
      terminating char
93     // return null pointer on the failiure of memory allocation
94     if (output == nullptr) {throw PassGen::exceptions::
      memoryAllocationFailiure(); return nullptr;}
95
96     for (int i = 0; i <= len; i++) {
97         if (i == len) {output[i] = termChar;} // ends with
      terminating char
98         else { // adds other chars otherwise
99
100             // adds random character from charList
101             randomCharPos = std::rand()%charListSize;
102             output[i] = charList[randomCharPos];
103
104             // set current and previous letter for check
105             currentLetter = output[i];
106             previousLetter = output[i-1];
107
108             // checks escape sequences which causes issues
109             while (previousLetter == backSlash &&
110                 ((currentLetter == 'a') || // '\a'
111                  (currentLetter == 'b') || // '\b'
112                  (currentLetter == 'c') || // '\c'
113                  (currentLetter == 'n') || // '\n'
114                  (currentLetter == 'f') || // '\f'
115                  (currentLetter == 'r') || // '\r'
116                  (currentLetter == 't') || // '\t'
117                  (currentLetter == 'U') || // '\U'
118                  (currentLetter == 'u') || // '\u'
119                  (currentLetter == 'v') || // '\v'
120                  (currentLetter == 'x') // '\x'
121                )) {
122                 // regenerate random letter
123                 randomCharPos = std::rand()%charListSize;
124                 output[i] = charList[randomCharPos];
125
126                 // re-set current letter
127                 currentLetter = output[i];
128             }}
129     }
130
131     return output;
132 }

```

utils.hxx

```

1  #ifndef UTILS_HXX
2  #define UTILS_HXX

```

```

3
4 #include <iostream>
5
6 /**
7  * @brief return the size(length) of string (pure C char list)
8  * @param str pure C string to be measured
9  * @return (int) - size(length) of string
10 */
11 inline auto strSize(const char *str) -> int {
12     int out = 0;
13     int index = 0;
14     while (str[index] != 0) {
15         index++;
16         out++;
17     }
18     return out;
19 }
20
21 /**
22  * @brief print the passed in argument
23  * @param object takes any type of input that is able to stdout to
24     the console
25  * @return (void) - console output of the object
26 */
27 template<class T>
28 inline auto printLine(T object) -> void {
29     std::cout << object << std::endl;
30 }
31
32 /**
33  * @brief prints the help for the console application
34  * @return (void) - console output of help document
35 */
36 inline auto printHelp() -> void {
37     printLine("APCSPCreateTask - Random Password Generator\n");
38     printLine("[Usage]: APCSPCreateTask [-A -a -n -s -g] -l <length>\n");
39     printLine("[Options]:\n");
40     printLine("\t-A, --upper : include upper case alphabets in password\n");
41     printLine("\t-a, --lower : include lower case alphabets in password\n");
42     printLine("\t-n, --number : include numbers in password\n");
43     printLine("\t-s, --special : include special characters in password\n");
44     printLine("\t-l, --length <number> : set the length of the password\n");
45     printLine("\t-g, --gui : run in GUI regardless of the previous options\n");
46     printLine("\t-h, --help : print this help\n");
47 }
48
49 /**
50  * @brief checks if any flag is enabled
51  * @param upper boolean flag for upper case letters
52  * @param lower boolean flag for lower case letters
53  * @param num boolean flag for numbers

```

```

53 * @param special boolean flag for special characters
54 * @return (bool) - returns true if one of any flag is enabled
55 */
56 inline auto checkFlags(bool upper, bool lower, bool num, bool
    special) -> bool {
57     bool isEnabled = (upper || lower || num || special);
58     return isEnabled;
59 }
60
61 #endif // UTILS_HXX

```

ui.hxx

```

1 #ifndef UI_HXX
2 #define UI_HXX
3
4 // enumeration of states of program
5 enum PROGRAMSTATE : int {
6     SUCCESS, // 0
7     FAIL // 1
8 };
9
10 // include libraries for string operations
11 #include <string>
12 #include <cstring>
13 // include utilities, passgen functions, and GUI app
14 #include <utils.hxx>
15 #include <passgen.hxx>
16 #include <gtkui.hxx>
17
18 /**
19 * @brief runs the CUI version of program
20 * @param len length of the password (int)
21 * @param upper boolean flag for upper case alphabets
22 * @param lower boolean flag for lower case alphabets
23 * @param num boolean flag for numbers
24 * @param special boolean flag for special characters
25 * @return (int) - execution state of program
26 */
27 inline auto runcui(const unsigned int& len, const bool& upper,
    const bool& lower, const bool& num, const bool& special) -> int
    {
28     // appends letters to input
29     std::string input;
30     try {
31         if (upper) {input += PassGen::getUpperAlpha();}
32         if (lower) {input += PassGen::getLowerAlpha();}
33         if (num) {input += PassGen::getNumber();}
34         if (special) {input += PassGen::getSpecialChars();}
35     }
36     // print out error when any of getChars threw exception
37     catch (PassGen::exceptions::memoryAllocationFailure &err) {
38         printLine(err.what());
39         return FAIL;
40     }
41
42     // converts to std::string to pure C string

```



```

43     char *cInput = new char[input.length() + 1]; // allocate memory
44     strcpy(cInput, input.c_str()); // copy the input to cInput
45
46     // generate password, report error and exit with status of 1 (
47     // failiure) if any caught
48     char *out = nullptr; // initialize with nullptr
49     try {
50         out = PassGen::passGen(cInput, len); // generate password
51     }
52     // print out error when PassGen::passGen threw exception
53     catch (PassGen::exceptions::memoryAllocationFailiure &err) {
54         printLine(err.what());
55         return FAIL;
56     }
57     catch (PassGen::exceptions::noCharList &err) {
58         printLine(err.what());
59         return FAIL;
60     }
61     catch (...) {
62         PassGen::exceptions::exception error = PassGen::exceptions
63         ::unknownError();
64         printLine(error.what());
65         return FAIL;
66     }
67
68     // prints password, and exit with status of 0 (success)
69     std::cout << out << std::endl;
70     return SUCCESS;
71 }
72
73 /**
74  * @brief runs the GUI version of program
75  * @return (int) - execution state of program
76  */
77 inline auto rungui() -> int {
78     // run the GTK application
79     auto app = Gtk::Application::create("apcsp.passgen"); // create
80     instance of application
81     return app->make_window_and_run<PassGenUI>(0, nullptr); // run
82     GTK app with no arguments
83 }
84
85 #endif // UI_HXX

```

gtkui.hxx

```

1  #ifndef GTKUI_HXX
2  #define GTKUI_HXX
3
4  #include <gtkmm.h> // GTK GUI Library (C++ wrapper)
5  #include <passgen.hxx>
6
7  class PassGenUI : public Gtk::Window
8  {
9  public:
10     PassGenUI(); // constructor

```

```

11     ~PassGenUI() override; // destructor
12
13     /**
14      * @brief The button event for m_generate_button
15      * @return (void) Generate password, set to m_output_buffer,
      and show to the user
16      */
17     auto on_generate_button_clicked() -> void; // button event
18
19     /**
20      * @brief Show error dialog with exceptions and message
21      * @param err PassGen::exceptions::exception exception to be
      reported
22      * @param extraMsg Glib::ustring Extra message to be shown
      along side the reported exception, set empty by default
23      * @return (void) Show the dialog
24      */
25     auto showErrorDialog(PassGen::exceptions::exception &err,
      Glib::ustring extraMsg) -> void;
26
27 private:
28     const int winHeight = 480;
29     const int winWidth = 640;
30     const int widgetMargin = 10;
31     const int maxLength = 8192;
32
33     // checkboxes
34     Gtk::CheckButton m_upper_check, m_lower_check, m_num_check,
      m_special_chars_check;
35
36     // boxes (containers)
37     Gtk::Box m_char_checks, m_output_box, m_main_box;
38
39     // title of the program
40     Gtk::Label m_title;
41
42     Gtk::Button m_generate_button;
43
44     // length input
45     Gtk::SpinButton m_num_input;
46
47     // sets range for m_num_input (0-maxLength)
48     Glib::RefPtr<Gtk::Adjustment> m_num_input_adj = Gtk::
      Adjustment::create(0, 0, maxLength);
49
50     // Provides CSS to the GTK widget
51     Glib::RefPtr<Gtk::CssProvider> m_output_style = Gtk::
      CssProvider::create();
52
53     // Text area for output
54     Gtk::ScrolledWindow m_output_scroll;
55     Gtk::TextView m_output;
56
57     // text buffer for m_output
58     Glib::RefPtr<Gtk::TextBuffer> m_output_buffer = Gtk::
      TextBuffer::create();
59 };

```

```

60
61 #endif // GTKUI_HXX

gtkui.cxx

1 #include "passgen.hxx"
2 #include "utils.hxx"
3 #include <gtkui.hxx>
4
5 PassGenUI::PassGenUI():
6     // initialize widgets
7     m_generate_button("Generate"),
8     m_main_box(Gtk::Orientation::VERTICAL, widgetMargin),
9     m_char_checks(Gtk::Orientation::VERTICAL, widgetMargin),
10    m_output_box(Gtk::Orientation::VERTICAL, widgetMargin),
11    m_upper_check("Include Upper Case Letters"),
12    m_lower_check("Include Lower Case Letters"),
13    m_num_check("Include Numbers"),
14    m_special_chars_check("Include Special Characters"),
15    m_title("Password Generator")
16 {
17     // set window props
18     set_title("AP CSP Create Task - Password Generator");
19     set_default_size(winWidth, winHeight);
20     // link the button event to the function
21     m_generate_button.signal_clicked().connect(sigc::mem_fun(*this,
22         &PassGenUI::on_generate_button_clicked));
23
24     // populate the widgets and other boxes in main box
25     set_child(m_main_box);
26     m_main_box.set_margin(widgetMargin);
27     m_num_input.set_adjustment(m_num_input_adj); // set the range
28     // of m_num_input widget can handle
29     m_main_box.append(m_title);
30     m_main_box.append(m_char_checks);
31     m_main_box.append(m_num_input);
32     m_main_box.append(m_generate_button);
33     m_main_box.append(m_output_box);
34
35     // populate the widgets in character configuration section
36     m_char_checks.append(m_upper_check);
37     m_char_checks.append(m_lower_check);
38     m_char_checks.append(m_num_check);
39     m_char_checks.append(m_special_chars_check);
40
41     // populate the widgets in output section
42     m_output_scroll.set_child(m_output);
43     m_output_scroll.set_expand();
44     // set the style of m_output using properties and CSS
45     m_output.set_editable(false);
46     m_output.set_monospace(true);
47     m_output.set_cursor_visible(false);
48     m_output_style->load_from_data("#m_output {font-size: 14pt;}");
49     m_output.set_name("m_output");
50     m_output.get_style_context()->add_provider(m_output_style, 1);
51     m_output.set_wrap_mode(Gtk::WrapMode::CHAR);
52     // add widget to the box

```

```

51     m_output_box.append(m_output_scroll);
52 }
53
54 PassGenUI::~PassGenUI() = default;
55
56 auto PassGenUI::showErrorDialog(PassGen::exceptions::exception &err
57     , Glib::ustring extraMsg="") -> void {
58     Glib::RefPtr<Gtk::AlertDialog> m_Alert = Gtk::AlertDialog::
59     create();
60     m_Alert->set_message("Error!");
61     m_Alert->set_detail((Glib::ustring)err.what() + "\n" + extraMsg
62 );
63     m_Alert->show(*this);
64 }
65
66 auto PassGenUI::on_generate_button_clicked() -> void {
67     // appends letters to input according to the flags
68     std::string input;
69     try {
70         if (m_upper_check.get_active()) {input += PassGen::
71         getUpperAlpha();}
72         if (m_lower_check.get_active()) {input += PassGen::
73         getLowerAlpha();}
74         if (m_num_check.get_active()) {input += PassGen::getNumber
75         ();}
76         if (m_special_chars_check.get_active()) {input += PassGen::
77         getSpecialChars();}
78     }
79     // show error dialog when any of getChars threw exception
80     catch (PassGen::exceptions::memoryAllocationFailure &err) {
81         showErrorDialog(err, "Please free some memory.");
82         return;
83     }
84
85     // convert std::string to pure C string
86     char* cInput = new char[input.length() + 1];
87     strcpy(cInput, input.c_str());
88
89     // get the length of password to be generated
90     int len = m_num_input.get_value_as_int();
91
92     // generate password
93     char* passwd = nullptr;
94     try {
95         passwd = PassGen::passGen(cInput, len);
96     }
97     // Show error dialog when PassGen::passGen threw exceptions
98     catch (PassGen::exceptions::memoryAllocationFailure &err) {
99         showErrorDialog(err, "Please free some memory.");
100         return;
101     }
102     catch (PassGen::exceptions::noCharList &err) {
103         showErrorDialog(err, "Please check the form.");
104         return;
105     }
106     catch (...) {
107         PassGen::exceptions::exception error = PassGen::exceptions

```

```

101         ::unknownError();
102         showErrorDialog(error);
103         return;
104     }
105
106     // Show the passwd by setting text and buffer
107     Glib::ustring output = Glib::convert(passwd, "UTF-8", "ISO
-8859-1"); // convert to appropriate type and encoding of text
108     m_output_buffer->set_text(output);
109     m_output.set_buffer(m_output_buffer);
110 }

```